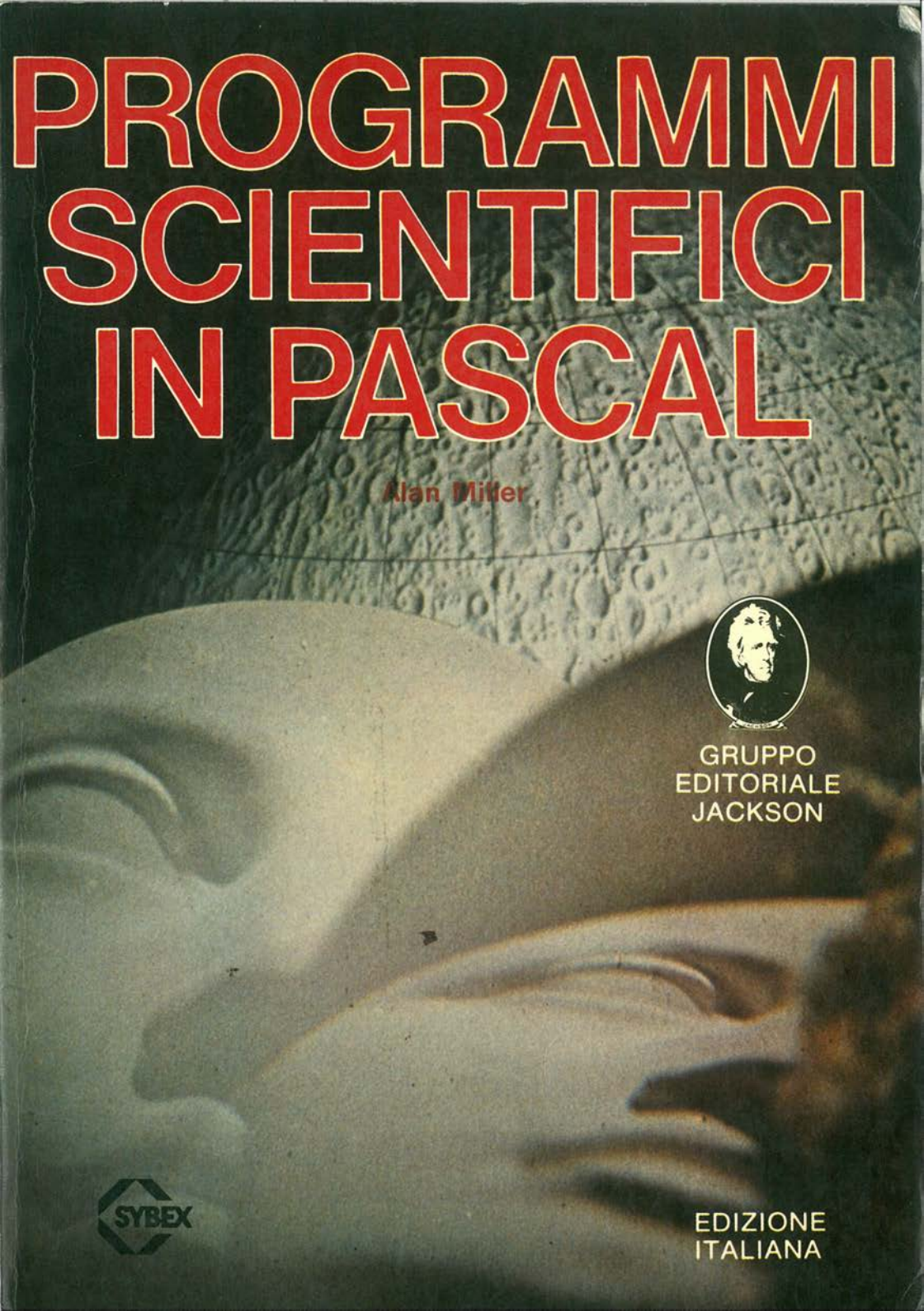
PROGRAMMI SCIENTIFICI IN PASCAL

Alan Miller



GRUPPO
EDITORIALE
JACKSON

SYBEX

EDIZIONE
ITALIANA

PROGRAMMI SCIENTIFICI IN PASCAL

di
Alan Miller

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

© Copyright per l'edizione originale Sybex Inc. 1981
© Copyright per l'edizione italiana Sybex Inc. 1983

Il Gruppo Editoriale Jackson ringrazia per il prezioso lavoro svolto nella stesura dell'edizione italiana la signora Francesca Di Fiore, e l'ing. Roberto Pancaldi.

Tutti i diritti sono riservati. Stampato in Italia. Nessuna parte di questo libro può essere riprodotta, memorizzata in sistemi di archivio, o trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopia, registrazione o altri senza la preventiva autorizzazione scritta dell'editore.

Stampato in Italia da:
S.p.A. Alberto Matarelli - Milano - Stabilimento Grafico

Fotocomposizione CorpoNove - Bergamo - tel. (035) 22.33.65

SOMMARIO

PREFAZIONE XII
INTRODUZIONE XIII
NOTA TIPOGRAFICA XIV

CAPITOLO 1 — VALUTAZIONE DI UN COMPILATORE PASCAL
Introduzione 1
Precisione e campo di applicazione delle operazioni in floating point 1
Programma Pascal: un test sulle operazioni in floating point 2
Funzioni Pascal SIN e COS 4
Programma Pascal: prova della funzione SIN 5
Altre funzioni Pascal 7
File 11
Una funzione potenza del 10 13
Programma Pascal: calcolo di una potenza del 10 15
Sommario 20

CAPITOLO 2 — MEDIA E DEVIAZIONE STANDARD
Introduzione 21
La media 21
La deviazione standard 23
Programma Pascal: media e deviazione standard 25
Numeri casuali 28
Funzione Pascal: generazione di numeri casuali 28
Programma Pascal: valutazione delle funzioni di generazione di numeri casuali 30
Funzione Pascal: distribuzione Gaussiana di numeri casuali 33
Programma Pascal: valutazione della funzione RANDG 35
Sommario 36

CAPITOLO 3 — OPERAZIONI SU VETTORI E MATRICI
Introduzione 37
Scalari e matrici 37
Vettori 38
Matrici 42
Programma Pascal: moltiplicazione tra matrici 47

Determinanti 51
Programma Pascal: determinanti 52
Matrice inversa e divisione tra matrici 55
Sommario 56

CAPITOLO 4 — RISOLUZIONE DI SISTEMI DI EQUAZIONI LINEARI

Introduzione 57
Equazioni lineari e risoluzione di sistemi 57
La regola di Cramer 59
Programma Pascal: un elegante uso della regola di Cramer 63
Il metodo di eliminazione di Gauss 64
Programma Pascal: il metodo di eliminazione di Gauss 71
Il metodo di eliminazione di Gauss-Jordan 78
Programma Pascal: il metodo di eliminazione di Gauss-Jordan 80
Vettori di termini noti multipli e inversione delle matrici 89
Programma Pascal: seconda versione del metodo di Gauss-Jordan 90
Equazioni mal condizionate 100
Programma Pascal: soluzione delle matrici di Hilbert 101
Miglior curva interpolante per un sistema di equazioni 105
Programma Pascal: miglior curva interpolante 106
Equazioni con coefficienti complessi 110
Programma Pascal: sistemi di equazioni con coefficienti complessi 114
Il metodo iterativo di Gauss-Seidel 120
Programma Pascal: il metodo di Gauss-Seidel 122
Sommario 129

CAPITOLO 5 — PROGRAMMA DI INTERPOLAZIONE DI UNA CURVA

Introduzione 131
Il programma principale 132
Una routine di stampa di un grafico 140
Simulazione di una routine di interpolazione di una curva 147
L'algoritmo di interpolazione di una curva 149
Il coefficiente di correlazione 155
Programma Pascal: interpolazione di una curva con i minimi quadrati per un insieme di dati simulati 157
Sommario 162

CAPITOLO 6 — ORDINAMENTO

Introduzione 163
Trattamento di dati sperimentali 163
Ordinamento a bolla 164
Programma Pascal: ordinamento a bolla e programma TSTSORT . . . 165
Procedura Pascal: ordinamento a bolla con scambio 169

Ordinamento di Shell 171
Procedura Pascal: ordinamento di Shell-Metzner 171
Ordinamento veloce 173
Procedura Pascal: una routine ricorsiva di ordinamento veloce 173
Procedura Pascal: una routine non ricorsiva di ordinamento veloce . 176
Aggiunta della routine di ordinamento nel programma di interpolazione di una curva 176
Sommario 180

CAPITOLO 7 — INTERPOLAZIONE DI UNA CURVA CON IL METODO DEI MINIMI QUADRATI

Introduzione 227
Interpolazione di una parabola 181
Interpolazione di una parabola 181
Programma Pascal: interpolazione di una parabola con i minimi quadrati 182
Interpolazione di curve per altre equazioni 189
Programma Pascal: l'approccio per matrici all'interpolazione di curve 193
Programma Pascal: determinazione del grado del polinomio 200
Programma Pascal: l'equazione della capacità termica 208
Programma Pascal: l'equazione della pressione di un vapore 211
Una equazione a tre variabili 216
Programma Pascal: un'equazione di stato per il vapore 217
Sommario 225

CAPITOLO 8 — RISOLUZIONE DI EQUAZIONI CON IL METODO DI NEWTON

Introduzione 227
Formulazione matematica del metodo di Newton 227
Programma Pascal: prima applicazione del metodo di Newton 233
Programmi Pascal: risoluzione di altre equazioni 242
Programma Pascal: l'equazione della pressione di un vapore 247
Sommario 248

CAPITOLO 9 — INTEGRAZIONE NUMERICA

Introduzione 249
L'integrale definito 249
La regola trapezoidale 251
Programma Pascal: la regola trapezoidale con il numero di trapezi introdotto come dato di ingresso dall'utente 253
Programma Pascal: una regola trapezoidale perfezionata 255
Programma Pascal: regola trapezoidale con correzione finale 259
Programma Pascal: metodo di integrazione di Simpson 262
Programma Pascal: il metodo di Simpson con correzione finale 267
Il metodo di Romberg 270

Programma Pascal: integrazione con il metodo di Romberg 271
Funzioni che tendono all'infinito ad un estremo dell'intervallo di integrazione 276
Programmma Pascal: sezioni di ampiezza variabile per una funzione infinita 276
Sommario 281

CAPITOLO 10 — EQUAZIONI NON LINEARI DI INTERPOLAZIONE DI CURVE
Introduzione 283
Linearizzazione della funzione razionale 283
Programma Pascal: il fattore di Clausing interpolato con la funzione razionale. 284
Linearizzazione dell'equazione esponenziale 291
Programma Pascal: un'interpolazione con la curva esponenziale per la diffusione dello zinco nel rame 291
Soluzione diretta dell'equazione esponenziale 296
Programma Pascal: interpolazione non lineare della curva esponenziale 299
Sommario 306

CAPITOLO 11 — APPLICAZIONI AVANZATE: LA CURVA NORMALE, LA FUNZIONE DI ERRORE GAUSSIANA, LA FUNZIONE GAMMA, LA FUNZIONE DI BESSEL
Introduzione 307
Le funzioni di distribuzione normale e cumulativa 307
La funzione di errore Gaussiana 310
Programma Pascal: calcolo della funzione di errore Gaussiana con la regola di Simpson 313
Programma Pascal: calcolo della funzione di errore Gaussiana con uno sviluppo in serie di infiniti termini 316
Il complemento della funzione di errore 319
Programma Pascal: calcolo del complemento della funzione di errore 319
Programma Pascal: un'implementazione più veloce del calcolo della funzione di errore 323
La funzione Gamma 326
Programma Pascal: calcolo della funzione Gamma 328
Funzioni di Bessel 331
Programma Pascal: funzioni di Bessel del primo tipo 333
Programma Pascal: funzioni di Bessel del secondo tipo 336
Sommario 341
Appendice A: Parole riservate e funzioni di sistema 343
Appendice B: Sommario del linguaggio Pascal 345
Insieme minimo dei caratteri standard 345
Nomi delle variabili 345

Numeri 346
Commenti 346
Operazioni 346
Sintassi 347
Istruzioni condizionali 350
Istruzioni iterative 352
Istruzioni di trasferimento del controllo 353
Ingresso e uscita 353
Tipi di dati 355
Indice

PROGRAMMI E ILLUSTRAZIONI

CAPITOLO 1

1.1 Un test per operazioni floating point 3
1.2 Test di precisione: primo compilatore 3
1.3 Test di precisione: secondo compilatore 4
1.4 Test di precisione: terzo compilatore 5
1.5 Prova della funzione SIN 6
1.6 Prova della funzione SIN: primo compilatore 6
1.7 Prova della funzione SIN: secondo compilatore 7
1.8 Prova delle funzioni LN e EXP 8
1.9 Funzione Arcotangente con due argomenti 9
1.10 La funzione Arcoseno 10
1.11 La funzione Arco coseno 10
1.12 Programma di prova delle funzioni Arcoseno e Arcotangente . . 11
1.13 Una funzione per calcolare le potenze del 10 17
1.14 Risultati del programma di calcolo delle potenze del 10 20

CAPITOLO 2

2.1 Una curva a forma di Gaussiana 22
2.2 Una maggiore dispersione 23
2.3 Calcolo della media e della deviazione standard 26
2.4 Risultati del programma di calcolo della media e della deviazione standard 28
2.5 Una funzione Pascal per generare numeri casuali 29
2.6 Risultati del programma RANTST 30
2.7 Programma di prova della funzione di generazione dei numeri casuali 32-33
2.8 Una funzione di generazione di numeri casuali secondo la distribuzione normale 34
2.9 Programma di prova della funzione Gaussiana di generazione di numeri casuali 35

CAPITOLO 3

3.1 Programma di moltiplicazione tra matrici ($A = X^T X$, $G = Y X$) 47
3.2 Risultati del programma di moltiplicazione tra matrici 51
3.3 Determinante di una matrice 3×3 53

CAPITOLO 4

4.1 Un circuito con resistenze e generatori di tensione continua . . 61
4.2 Risoluzione di un sistema di tre equazioni lineari con la regola di Cramer 65
4.3 Risoluzione di sistemi con il metodo di eliminazione di Gauss . 72
4.4 Risultati del programma di risoluzione del circuito elettrico con il metodo di eliminazione di Gauss 77
4.5 Risoluzione di sistemi di equazioni con il metodo di eliminazione di Gauss-Jordan 81
4.6 Procedura di Gauss-Jordan 84
4.7 Risultati del secondo programma del metodo di Gauss-Jordan . 91
4.8 Risoluzione di sistemi e inversione di matrici con il metodo di Gauss-Jordan 92
4.9 Risoluzione di un sistema di equazioni mal condizionate 102
4.10 Miglior interpolante bidimensionale 105
4.11 Miglior interpolante per un sistema di equazioni lineari 106
4.12 Circuito con generatore di tensione alternata 111
4.13 Risoluzione di sistemi di equazioni con coefficienti complessi . . 114
4.14 Risoluzione di equazioni lineari con il metodo di Gauss-Seidel . 123

CAPITOLO 5

5.1 Parte iniziale di un programma di interpolazione di una curva . . 133
5.2 Routine alternativa con la procedura RANDOM 136
5.3 Prima esecuzione del programma di interpolazione di una curva 140
5.4 Procedura di stampa di un grafico 141
5.5 Grafico di y in funzione di x 146
5.6 Procedura LINFIT di simulazione di un'interpolazione lineare . . 147
5.7 La procedura WRITE-DATA modificata 148
5.8 Programma principale 148
5.9 Grafico di y e Y-CALC in funzione di x 149
5.10 Procedura LINFIT per interpolazione con i minimi quadrati . . . 153
5.11 Interpolazione con i minimi quadrati 155
5.12 Quando non c'è correlazione tra x e y 156
5.13 Dati molto dispersi 156
5.14 Il programma completo di interpolazione di una curva 158

CAPITOLO 6

6.1 Una procedura di ordinamento a bolla con programma principale 166
6.2 Una variante dell'ordinamento a bolla, con procedura separata SWAP 170
6.3 Una procedura di ordinamento di Shell 171
6.4 Una versione ricorsiva dell'ordinamento veloce 174
6.5 Una versione non ricorsiva dell'ordinamento veloce 175

CAPITOLO 7

7.1 Interpolazione parabolica con i minimi quadrati 183
7.2 Risultati del programma di interpolazione parabolica 190
7.3 Interpolazione parabolica con i minimi quadrati con il metodo di eliminazione di Gauss-Jordan 194
7.4 Risultati della versione alternativa dell'interpolazione parabolica con i minimi quadrati 200
7.5 Introduzione dell'ordine del polinomio dalla console 201
7.6 Interpolazione rettilinea per dati parabolici 207
7.7 Procedure GET-DATA e LINFIT per l'equazione della capacità termica 208
7.8 Risultati relativi alla capacità termica dell'ossigeno 211
7.9 Procedure GET-DATA e LINFIT per l'equazione della pressione di un vapore 212
7.10 Risultati relativi alla pressione del vapore del piombo 215
7.11 Un'equazione di stato per il vapore 218
7.12 Proprietà del vapore surriscaldato (n = 1) 222
7.13 Proprietà del vapore surriscaldato (n = 2) 223
7.14 Proprietà del vapore surriscaldato (n = 3) 223
7.15 Proprietà del vapore surriscaldato (n = 4) 224

CAPITOLO 8

8.1 Una funzione con due soluzioni 229
8.2 Una funzione con una soluzione 230
8.3 Una funzione senza radici reali 230
8.4 $f(x) = 0$ dove la curva incontra l'asse x 231
8.5 La tangente incrocia l'asse delle xx più vicino alla radice della prima approssimazione per x 232
8.6 Metodo di Newton: prima versione 234
8.7 La radice positiva di $f(x) = x^2 - 2$ 236
8.8 Il programma principale della seconda versione 237
8.9 Dove $f'(x) = 0$ la tangente è parallela all'asse x 239
8.10 Il metodo di Newton con un test per tangente 0 240
8.11 Una radice complessa 241
8.12 Metodo di Newton con un contatore di ciclo 243
8.13 Procedura FUNC per $e^x = 4x$ 244
8.14 Soluzione di $\sin(x) = x/10$ 245
8.15 Procedura alternativa FUNC per $\sin(x) = x/10$ 246
8.16 Le radici di $f(x) = \sin x - x/10$ 247
8.17 Soluzione dell'equazione della pressione del vapore 248

CAPITOLO 9

9.1 L'area sottesa dalla curva $y = f(x)$ 250

9.2 Calcolo dell'area con una sola sezione 252
9.3 La regola trapezoidale 253
9.4 Un metodo trapezoidale perfezionato 256
9.5 Integrazione con il metodo trapezoidale perfezionato 258
9.6 Funzione DFX e procedura TRAPEZ con correzione finale 260
9.7 Integrazione trapezoidale con correzione finale 261
9.8 Regola di Simpson 263
9.9 Integrazione con la regola di Simpson 265
9.10 Integrazione di e^{-x^2} con la regola di Simpson 266
9.11 L'integrale di $\sin^2 x$ tra 0 e 4 267
9.12 Procedura SIMPS con correzione finale 268
9.13 Integrazione con la regola di Simpson con correzione finale . . 269
9.14 Il metodo di Romberg 271
9.15 Integrazione con il metodo di Romberg 274
9.16 Integrazione di e^{-x^2} con il metodo di Romberg 275
9.17 Integrazione di $\sin^2 x$ con il metodo di Romberg 277
9.18 Integrazione con il metodo di Romberg con sezioni di ampiezza variabile 277
9.19 Integrazione di $1/\sqrt{x}$ vicino allo zero 281

CAPITOLO 10

10.1 Il fattore di Clausing interpolato con il rapporto tra due polinomi 285
10.2 Il fattore di Clausing in funzione di L/r interpolato con una funzione razionale 290
10.3 Un'interpolazione con i minimi quadrati per un'equazione esponenziale linearizzata 292
10.4 La diffusione dello zinco nel rame (interpolazione lineare) . . . 296
10.5 Interpolazione con i minimi quadrati per una funzione esponenziale non linearizzata 300
10.6 La diffusione dello zinco nel rame (interpolazione non lineare) 306

CAPITOLO 11

11.1 La funzione normale di distribuzione della probabilità 308
11.2 Curve normali con deviazione standard grande (a) e piccola (b) 309
11.3 La funzione di distribuzione cumulativa 310
11.4 La funzione di errore Gaussiana per la regola di Simpson . . . 313
11.5 Uno sviluppo in serie infinita per la funzione di errore Gaussiana 316
11.6 La funzione di errore Gaussiana 318
11.7 La funzione di errore e il suo complemento 320
11.8 La funzione di errore complementare 323
11.9 Un calcolo non iterativo della funzione di errore 324
11.10 La funzione Gamma 328
11.11 Alcuni valori della funzione Gamma 329

11.12 Valutazione della funzione Gamma 330
11.13 Le funzioni di Bessel J_0 e J_1 332
11.14 Alcuni valori della funzione di Bessel 333
11.15 La funzione di Bessel del primo tipo 334
11.16 Le funzioni di Bessel Y_0 e Y_1 337
11.17 Alcuni valori della funzione di Bessel del secondo tipo 338
11.18 La funzione di Bessel del secondo tipo 338

PREFAZIONE

Ho raccolto le idee e il materiale per questo libro durante il mio insegnamento di ingegneria. Negli ultimi 14 anni il linguaggio che insegnavo nelle mie classi è cambiato da FORTRAN a BASIC e, infine, a quello che forse è il più adatto per gli studenti, il PASCAL.

Tutti i programmi in PASCAL di questo libro sono stati sviluppati su un microcalcolatore Z-80 con sistema operativo CP/M versione Lifeboat 2.2. I programmi sorgente sono stati scritti sotto Text Editor Word-Master MicroPro, e compilati con il PASCAL/M. La maggior parte dei programmi girano anche sotto altri compilatori Pascal, come la versione CP/M del Pascal/MT+, Pascal/Z, Pascal JRT, e un compilatore Pascal del Dec-20.

Il manoscritto è stato prodotto con il MicroPro's WordStar, sullo stesso calcolatore Z-80, e i programmi sorgente in Pascal vi sono stati inseriti direttamente dai file su cui erano registrati. Le stampe eseguite con il calcolatore, che compaiono nelle figure, sono anch'esse state inserite magneticamente, modificando il sistema operativo CPM in modo che l'uscita della stampante venisse scritta in un blocco di memoria. La copia finale del manoscritto è stata consegnata alla SYBEX in una forma compatibile con il fotocompositore; quindi non è stato necessario ribattere nè il manoscritto nè i programmi sorgente.

Sono grato del contributo e dei suggerimenti di Rudolph e Douglas Hergert, e vorrei anche ringraziare Ashok Singh che ha rivisto il manoscritto, con particolare attenzione alle espressioni matematiche.

ALAN R. MILLER

INTRODUZIONE

L'obiettivo di questo libro è duplice: aiutare il lettore a perfezionarsi nell'uso del Pascal, e insieme costruire una libreria di programmi per risolvere i problemi che si incontrano più frequentemente nel campo scientifico e ingegneristico.

I programmi di questo libro si riveleranno particolarmente utili per gli esperti della materia, e possono essere usati in un corso di ingegneria di livello senior o junior. Il lettore deve avere una conoscenza pratica di un linguaggio come il Pascal, il FORTRAN o il BASIC e possibilmente la conoscenza delle operazioni sui vettori e del calcolo differenziale e integrale.

Le caratteristiche del Pascal consentono un tipo di programmazione chiaro ed elegante; due di queste sono particolarmente comode nei programmi scientifici: i nomi lunghi per le variabili e la struttura a blocchi. Quando si usano identificatori come SOMMA_X_QUADRO e NESSUN_CAMBIAMENTO invece di nomi brevi e meno significativi, il codice sorgente risulta più facile da capire. Il Pascal ha inoltre una vasta gamma di istruzioni iterative, molte più di quante ce ne siano in genere in BASIC e FORTRAN. L'istruzione

FOR I := 1 **TO** N **DO**

corrisponde al ciclo **FOR...NEXT** del BASIC e **DO...CONTINUE** del FORTRAN; in Pascal esistono altre costruzioni di tipo iterativo, come **WHILE...DO** e **REPEAT...UNTIL.**

Le caratteristiche fondamentali del Pascal sono riassunte nel "Pascal User Manual and Report", di Jensen e Wirth (1); ulteriori dettagli si trovano in "Introduction to Pascal", di Zaks (2). D'altra parte, non tutte le specifiche del Pascal sono previste nei compilatori reali; per esempio, la maggior parte dei compilatori Pascal non consente che i nomi delle procedure e delle funzioni siano usati come parametri di altre procedure. (Questo problema è affrontato nel Capitolo 8 sotto il titolo "Chiamate alle procedure").

(1) Kathleen Jensen e Niklans Wirth, Pascal User Manual and Report, Seconda Edizione (New York, Heidelberg, e Berlino: Springer - Verlag, 1974).
(2) Rodnay Zaks, Introduction to Pascal, Lucludring UCSD Pascal (Berkeley: Sybex, 1980).

Invece, diverse specifiche particolarmente utili, che mancano nella definizione originale del Pascal, sono state aggiunte nei compilatori in commercio, purtroppo non in modo uniforme; una di queste è il tipo di stringa dinamica; un'altra il generatore di numeri casuali, descritto nel Capitolo 2.

In questo libro talvolta vengono usate costruzioni comuni ad altri linguaggi evoluti, come il FORTRAN, il BASIC e l'ALGOL, al posto di quelle più eleganti del Pascal; ad esempio, si usano preferibilmente le matrici al posto dei record. Perciò gli algoritmi presentati nel testo, possono essere facilmente convertiti in altri linguaggi.

Chi è interessato soprattutto ai programmi di Pascal, non avrà problemi a trovarli; i paragrafi che contengono programmi, procedure e funzioni sono segnalati chiaramente. D'altra parte questo libro è fatto per essere letto dall'inizio alla fine; ogni capitolo affronta e sviluppa le conoscenze che serviranno poi nei capitoli successivi. Gli algoritmi matematici di ogni programma sono descritti sistematicamente prima di implementare il programma stesso, e per quasi tutti i programmi viene fornito un esempio dei risultati prodotti. Riassumiamo di seguito, brevemente, il contenuto dei singoli capitoli:

CAPITOLO 1: **Valutazione di un compilatore Pascal,** mette in evidenza i punti salienti di alcuni compilatori Pascal in commercio, e fornisce dei programmi per provare qualsiasi compilatore Pascal; i risultati di questi test saranno poi usati per scegliere costanti e procedure diverse nei capitoli successivi. Contiene anche una presentazione delle funzioni logaritmica, esponenziale e trigonometriche.

CAPITOLO 2: **Media e deviazione standard,** presenta alcuni classici algoritmi di statistica e un programma che li implementa; inoltre fornisce le procedure per generare e provare la distribuzione uniforme e quella Gaussiana dei numeri casuali.

CAPITOLO 3: **Operazioni su vettori e matrici,** presenta le operazioni aritmetiche su vettori e matrici, il prodotto scalare, il prodotto vettoriale, il prodotto tra matrici, e l'inversione di matrici. Contiene due importanti programmi, uno per eseguire la moltiplicazione tra matrici e uno per calcolare i determinanti.

CAPITOLO 4: **Risoluzione di sistemi di equazioni lineari,** presenta dei programmi che applicano la regola di Cramer il metodo di eliminazione di Gauss, il metodo di Gauss Jordan, il metodo di Gauss Seidel, alla risoluzione di sistemi di equazioni lineari. Viene affrontato il problema di sistemi mal condizionati con un programma che genera le matrici di Hilbert, ed è sviluppato un programma che risolve le equazioni a coefficienti complessi.

CAPITOLO 5: **Programma di interpolazione di una curva,** è il primo di una serie di capitoli sull'interpolazione di curve; presenta un programma di interpolazione lineare con i minimi quadrati, insieme con una buona illustrazione della tecnica di programmazione top - down. Il programma comprende procedure per simulare dati, stampare grafici, calcolare i valori della curva interpolata e fornire il coefficiente di correlazione.

CAPITOLO 6: **Ordinamento,** descrive e confronta diverse routine di ordinamento in Pascal, con due ordinamenti a bolle, un ordinamento di Shell, un ordinamento velo-

ce ricorsivo e uno non ricorsivo. Viene inserita una routine di ordinamento nel programma di interpolazione del Capitolo 5, per metterlo in grado di trattare dati reali sperimentali.

CAPITOLO 7: **Interpolazione di una curva,** estende il programma di interpolazione al caso più generale di equazioni polinomiali e risolve tre casi particolari: equazione della capacità termica, equazione della pressione del vapore e equazione di stato del vapore surriscaldato.

CAPITOLO 8: **Risoluzione di equazioni con il metodo di Newton,** presenta una serie di programmi che fanno uso del metodo di Newton per trovare le radici di un'equazione. Questa tecnica verrà usata in seguito nel Capitolo 10 per l'interpolazione non lineare.

CAPITOLO 9: **Integrazione numerica,** sviluppa i programmi per tre diversi metodi di integrazione, la regola trapezoidale, la regola di Simpson, e il metodo di Romberg; viene anche discussa la tecnica di correzione finale dell'errore. La regola di Simpson sarà usata nel Capitolo 11 per calcolare la Gaussiana dell'errore.

CAPITOLO 10: **Equazioni non lineari di interpolazione di curve,** discute l'algoritmo di interpolazione per la funzione razionale ed esponenziale; vengono dati due esempi, il fattore di Clausing e l'equazione di diffusione.

CAPITOLO 11: **Applicazioni avanzate:** la curva Normale, la funzione di errore Gaussiana, la funzione Gamma e la funzione di Bessel, affronta diversi e complessi problemi relativi alla programmazione di applicazioni matematiche. Riassume e approfondisce alcuni dei concetti presentati nei capitoli precedenti.

Per il lettore che affronta il Pascal per la prima volta, c'è in appendice, un riepilogo della sintassi, delle funzioni standard e delle parole riservate di Pascal. Ci auguriamo per altro che gli obiettivi didattici di questo libro si raggiungano attraverso un accurato studio dei programmi.

ʒ

CAPITOLO 1

VALUTAZIONE DI UN COMPILATORE PASCAL

INTRODUZIONE

Per capire i risultati di un programma in Pascal occorre conoscere bene i limiti del compilatore che usiamo. Questo è particolarmente vero per i programmi di applicazioni scientifiche, come quelli presentati in questo libro. In questo primo capitolo presenteremo alcuni strumenti per valutare la precisione e il campo di applicazione di un compilatore Pascal; in effetti gli esempi che verranno fatti sono ricavati da alcuni compilatori Pascal disponibili in commercio.

Nelle nostre valutazioni, considereremo un certo numero di funzioni di Pascal, logaritmiche, esponenziali e trigonometriche, e tramite queste scopriremo alcune carenze dei compilatori reali. Scopriremo anche il modo per provare le funzioni fornite dai tipici compilatori Pascal, e per realizzarne altre generalmente non disponibili.

PRECISAZIONE E CAMPO DI APPLICAZIONE DELLE OPERAZIONI IN FLOATING-POINT

La corretta esecuzione di molti programmi di questo libro dipende dalla precisione e dal campo di applicazione delle operazioni in floating-point in Pascal. Se, per esempio, in un programma, un algoritmo ha fine quando il valore di una variabile diventa inferiore ad una certa tolleranza, la formula che rappresenta questa situazione è

```
TERM < TOL * SUM
```

dove TERM contiene il valore della variabile, SUM è il totale e TOL è un numero piccolo a piacere, che prende il nome di tolleranza.

È importante che il valore scelto per la tolleranza sia compatibile con la precisione delle operazioni in floating-point, altrimenti la sommatoria non avrà mai termine. Supponiamo per esempio che si raggiunga una precisione di sei cifre significative; la tolleranza dovrà avere un valore maggiore di 10^{-6}.

Un altro problema è l'intervallo consentito per l'esponente. In genere le operazioni floating-point su numeri binari raggiungono 32 bit di precisione; sui numeri in BCD raggiungono una precisione maggiore.

Nel Paragrafo seguente presenteremo un programma che prova la precisione e il campo di applicazione di un compilatore Pascal; studieremo l'uscita di diversi compilatori per determinarne la precisione sia nella mantissa che nell'esponente.

PROGRAMMA PASCAL: UN TEST SULLE OPERAZIONI IN FLOATING POINT

Il programma della figura 1.1 può essere usato per determinare la precisione e il campo di applicazione di un compilatore Pascal. Scrivetelo ed eseguitelo. Il valore iniziale di X si ottiene dividendo 10^{-4} per 3; poi vengono calcolati valori sempre più piccoli di X e visualizzati sulla console. Ogni valore si ottiene dal prodotto di 0.1 per il valore precedente. Il procedimento continua fino a quando sono stati stampati 40 valori, o un errore nelle operazioni in floating-point interrompe il programma.

Usiamo ora questo programma per provare tre diversi compilatori.

Tre esecuzioni del programma: Un confronto

La mantissa iniziale è 1/3, numero periodico che non può essere rappresentato esattamente in floating-point. Le successive moltiplicazioni mostreranno un aumento dell'errore di arrotondamento. Un numero binario floating-point di 32 bit, in genere, usa tre byte per la mantissa e uno per l'esponente. In questo modo raggiunge sei o sette cifre significative di precisione, e un intervallo di valori compresi tra 10^{+38} e 10^{-38}. Il risultato si presenterà come in figura 1.2. Nel primo caso un errore in una operazione floating-point interrompe il programma quando la variabile assume un valore al limite dell'intervallo; la precisione della mantissa in quel momento è di circa cinque cifre significative; l'effetto dell'errore di arrotondamento è visibile nelle cifre meno significative.

Consideriamo ora l'uscita (figura 1.3) di un diverso compilatore Pascal, che usa ancora 32 bit per i numeri floating-point. In questo caso la mantissa mostra una precisione un po' minore, ma, cosa più importante, l'intervallo è solo da 10^{-18} a 10^{+18} Inoltre, l'under-flow che si verifica in un'operazione floating-point non interrompe il programma, e quindi produce un risultato privo di significato.

```
PROGRAM TEST(OUTPUT);
(* test sul range dei numeri floating point *)

VAR
   I : INTEGER;
   X : REAL;

BEGIN
   WRITELN;
   X := 1.0E-4/3.0;
   FOR I := 1 TO 40 DO
      BEGIN
         WRITE(' x =', X);
         X := 0.1 * X;
         WRITELN('  x =', X);
         X := 0.1 * X
      END
END.
```

Figura 1.1 — Un test per operazioni floating-point.

```
x = 3.33333E-5    x = 3.33333E-6
x = 3.33333E-7    x = 3.33333E-8
x = 3.33333E-9    x = 3.33333E-10
x = 3.33333E-11   x = 3.33333E-12
x = 3.33333E-13   x = 3.33333E-14
x = 3.33333E-15   x = 3.33333E-16
x = 3.33333E-17   x = 3.33333E-18
x = 3.33333E-19   x = 3.33333E-20
x = 3.33333E-21   x = 3.33333E-22
x = 3.33333E-23   x = 3.33333E-24
x = 3.33333E-25   x = 3.33333E-26
x = 3.33333E-27   x = 3.33334E-28
x = 3.33334E-29   x = 3.33334E-30
x = 3.33334E-31   x = 3.33334E-32
x = 3.33334E-33   x = 3.33334E-34
x = 3.33334E-35   x = 3.33334E-36
x = 3.33334E-37   x = 3.33334E-38
x =
Floating point error  (messaggio di errore dal compilatore)
```

Figura 1.2 — Test di precisione: primo compilatore.

```
x = .3333334E-04   x = .3333333E-05
x = .3333332E-06   x = .3333330E-07
x = .3333330E-08   x = .3333329E-09
x = .3333328E-10   x = .3333327E-11
x = .3333327E-12   x = .3333325E-13
x = .3333325E-14   x = .3333325E-15
x = .3333323E-16   x = .3333323E-17
x = .3333323E-18   x = .0000000E-18
x = .1229780E+19   x = .1134271E+18
x = .1134271E+17   x = .1134271E+16
...
```

Figura 1.3 — Test di precisione: secondo compilatore.

Nella figura 1.4 c'è l'uscita prodotta da un terzo compilatore Pascal che usa 64 bit e numeri in BCD; la mantissa ha 14 cifre di precisione e l'intervallo è $10^{+63} - 10^{-63}$. Notate che anche in questo caso, come in quello precedente, un errore di un'operazione in floating-point non interrompe il programma.

Adesso vedremo una carenza più interessante, comune a molti compilatori, che limita l'intervallo dei valori per cui si possono calcolare funzioni SIN e COS, e approfondiremo questo argomento.

FUNZIONI PASCAL SIN E COS

Quando l'argomento delle funzioni SIN e COS si avvicina a zero, può presentarsi un problema: quando l'ampiezza dell'argomento è minore di 10^{-8}, il SIN dovrebbe restituire un valore pari all'argomento, e il COS l'unità. Invece, molti compilatori Pascal in commercio commettono il seguente errore: l'argomento viene elevato al quadrato prima di eseguire il test sull'intervallo; questo causa un underflow e i risultati prodotti sono senza significato.

Nella prossima sezione presenteremo un programma che studia questo problema e proveremo due compilatori.

```
x = +0.33333333333333E-04   x = +0.33333333333333E-05
x = +0.33333333333333E-06   x = +0.33333333333333E-07
x = +0.33333333333333E-08   x = +0.33333333333333E-09
x = +0.33333333333333E-10   x = +0.33333333333333E-11
x = +0.33333333333333E-12   x = +0.33333333333333E-13
x = +0.33333333333333E-14   x = +0.33333333333333E-15
x = +0.33333333333333E-16   x = +0.33333333333333E-17
x = +0.33333333333333E-18   x = +0.33333333333333E-19
x = +0.33333333333333E-20   x = +0.33333333333333E-21
x = +0.33333333333333E-22   x = +0.33333333333333E-23
x = +0.33333333333333E-24   x = +0.33333333333333E-25
x = +0.33333333333333E-26   x = +0.33333333333333E-27
x = +0.33333333333333E-28   x = +0.33333333333333E-29
x = +0.33333333333333E-30   x = +0.33333333333333E-31
x = +0.33333333333333E-32   x = +0.33333333333333E-33
x = +0.33333333333333E-34   x = +0.33333333333333E-35
x = +0.33333333333333E-36   x = +0.33333333333333E-37
x = +0.33333333333333E-38   x = +0.33333333333333E-39
x = +0.33333333333333E-40   x = +0.33333333333333E-41
x = +0.33333333333333E-42   x = +0.33333333333333E-43
x = +0.33333333333333E-44   x = +0.33333333333333E-45
x = +0.33333333333333E-46   x = +0.33333333333333E-47
x = +0.33333333333333E-48   x = +0.33333333333333E-49
x = +0.33333333333333E-50   x = +0.33333333333333E-51
x = +0.33333333333333E-52   x = +0.33333333333333E-53
x = +0.33333333333333E-54   x = +0.33333333333333E-55
x = +0.33333333333333E-56   x = +0.33333333333333E-57
x = +0.33333333333333E-58   x = +0.33333333333333E-59
x = +0.33333333333333E-60   x = +0.33333333333333E-61
x = +0.33333333333333E-62   x = +0.33333333333333E-63
x = +0.33333333333333E+00   x = -0.33333333333333E+63
x = -0.33333333333333E+62   x = -0.33333333333333E+61
```

Figura 1.4 — Test di precisione: terzo compilatore.

PROGRAMMA PASCAL: PROVA DELLA FUNZIONE SIN

Il programma della figura 1.5 può essere usato per provare la funzione SIN del vostro Pascal: scrivetelo ed eseguitelo.

Prova della funzione SIN: due Compilatori

Se la vostra funzione SIN tratta correttamente i piccoli numeri, dovrebbe restituire valori significativi in tutto l'intervallo delle operazioni in floating-point; quindi l'underflow dovrebbe capitare nello stesso momento del test precedente. Nella figura 1.6 i limiti dell'intervallo, sia per le operazioni floating-point che per la funzione SIN sono 3.3×10^{-18}.

```
PROGRAM TSIN(OUTPUT);
(* test sul range di sin *)

VAR
   I : INTEGER;
   X : REAL;

BEGIN
   X := 1.0E-4/0.3;
   FOR I := 1 TO 40 DO
      BEGIN
         WRITELN(' x =', X, ', sin =', SIN(X));
         X := 0.1 * X
      END
END.
```

Figura 1.5 — Prova della funzione SIN.

```
x =  3.333333E-04, sin =  3.333329E-04
x =  3.333332E-05, sin =  3.333328E-05
x =  3.333332E-06, sin =  3.333328E-06
x =  3.333332E-07, sin =  3.333328E-07
x =  3.333331E-08, sin =  3.333327E-08
x =  3.333330E-09, sin =  3.333326E-09
x =  3.333330E-10, sin =  3.333326E-10
x =  3.333329E-11, sin =  3.333325E-11
x =  3.333328E-12, sin =  3.333324E-12
x =  3.333328E-13, sin =  3.333324E-13
x =  3.333328E-14, sin =  3.333323E-14
x =  3.333327E-15, sin =  3.333322E-15
x =  3.333327E-16, sin =  3.333322E-16
x =  3.333326E-17, sin =  3.333322E-17
x =  3.333326E-18, sin =  3.333322E-18
Multiply overflow      (messaggio di errore dal compilatore)
```

Figura 1.6 — Prova della funzione SIN: primo compilatore.

Alcuni compilatori, nell'implementare la funzione SIN, eseguono il quadrato dell'argomento prima di effettuare un controllo sull'intervallo; in questo caso l'underflow si avrà molto prima. Nella figura 1.7 i limiti dell'intervallo sono 10^{+18} e 10^{-18}, ma si ha underflow durante il calcolo della funzione SIN quando l'argomento diventa minore di 10^{-9}.

```
x = .3333335E-03, sin = .3333323E-03
x = .3333334E-04, sin = .3333321E-04
x = .3333334E-05, sin = .3333321E-05
x = .3333333E-06, sin = .3333320E-06
x = .3333331E-07, sin = .3333319E-07
x = .3333331E-08, sin = .3333319E-08
x = .3333330E-09, sin = .3333318E-09
x = .3333329E-10, sin =-.2100363E+07
x = .3333328E-11, sin =-.2100361E+04
...
```

Figura 1.7 — Prova della funzione SIN: secondo compilatore.

ALTRE FUNZIONI PASCAL

In questa sezione studieremo dei programmi progettati per provare altre funzioni; impareremo anche a scrivere alcune funzioni trigonometriche non previste nei compilatori Pascal standard, e esamineremo a fondo la funzione ARCTAN.

In certi casi una funzione può essere associata alla sua inversa; ad esempio si può estrarre il LOG della funzione EXP, come si vede nella figura 1.8; oppure si può calcolare la funzione ARCTAN del rapporto SIN/COS. (La funzione TAN non esiste nel Pascal standard).

I compilatori Pascal in genere hanno le funzioni SQRT, SQR, EXP, LN, SIN, COS e ARCTAN. Purtroppo, la funzione ARCTAN per argomenti negativi non viene calcolata nello stesso modo in tutti i compilatori: alcuni restituiscono un angolo nel secondo quadrante, (da 90° a 180°), altri nel quarto (da 0° a −90°); inoltre non si possono distinguere gli angoli del primo quadrante da quelli del terzo.

```
PROGRAM TLOG(OUTPUT);
(* test ln ed exp *)

VAR
   I : INTEGER;
   X, Y : REAL;

BEGIN
   X := 1.0E-4/0.3;
   FOR I := 1 TO 20 DO
      BEGIN
         Y := LN(X);
         WRITELN(' x =', X, ', ln =', Y, ', exp(ln) =', EXP(Y));
         X := 0.5 * X
      END
END.
```

Figura 1.8 — Prova delle funzioni LN e EXP.

La funzione ARCTAN mostrata in Figura 1.9 tratta correttamente tutte queste possibilità. Vengono forniti due argomenti, la x e la y dell'angolo, e viene chiamata la funzione standard Pascal ARCTAN con un argomento positivo; in questo modo il risultato risulta corretto per ogni quadrante. Per esempio, consideriamo un angolo con x=—5 e y=—5, cioè un angolo di (180+45)° nel terzo quadrante. La tangente di questo angolo è 1, e la funzione standard ARCTAN darebbe un angolo di 45°; la funzione ATAN della figura 1.9 restituisce invece correttamente un risultato di 225°. Notate che questa funzione dà gli angoli in gradi invece che in radianti.

Nei compilatori Pascal standard non esistono le funzioni Arcoseno e Arco-coseno, ma possono essere ricavate dalla arcotangente. La funzione della figura 1.10 serve per calcolare l'arcoseno, con la funzione arcotangente della figura 1.9; l'arcoseno restituisce un angolo nel primo quadrante se l'argomento è positivo, un angolo negativo nel quarto quadrante se l'argomento è negativo. Il valore dell'angolo è in gradi.

```
FUNCTION ATAN(X, Y : REAL): REAL;
(* arctan in gradi *)
CONST
  PI180 = 57.2957795;
VAR
  A : REAL;
BEGIN (* function atan *)
  IF X = 0.0 THEN
    IF Y = 0.0 THEN ATAN := 0.0
    ELSE ATAN := 90.0
  ELSE (* x <> 0 *)
    IF Y = 0.0 THEN  ATAN := 0.0
    ELSE (* x e y <> 0 *)
      BEGIN
        A := ARCTAN(ABS(Y/X)) * PI180;
        IF X > 0.0 THEN
          IF Y > 0.0 THEN ATAN := A (* x, y > 0 *)
          ELSE ATAN := -A (* x > 0, y < 0 *)
        ELSE (* x < 0 *)
          IF Y > 0.0 THEN ATAN := 180.0 -A (* x < 0, y > 0 *)
          ELSE ATAN := 180.0 +A (* x, y < 0 *)
      END (* else *)
END (* function atan *);
```

Figura 1.9 — Funzione Arcotangente con due argomenti.

Nella figura 1.11 è mostrata la funzione arco-coseno, realizzata tramite la funzione arcotangente della figura 1.9. Se l'argomento è positivo, il risultato è un angolo nel primo quadrante; se è negativo, nel secondo quadrante.

Nella figura 1.12 è mostrato un programma per provare le funzioni "arco", con le funzioni ATAN della figura 1.9 e ARCSIN della figura 1.10.

In questo paragrafo abbiamo visto l'implementazione di funzioni logaritmiche e trigonometriche, sia fornite dai compilatori Pascal standard sia realizzate da utente. Ritorneremo a questo argomento alla fine del capitolo, dopo aver studiato le opzioni Pascal di gestione file.

```
FUNCTION ARCSIN(X : REAL): REAL;
(* arcsin in gradi *)
(* è richiesta la funzione ATAN *)
(* 1 Feb 81 *)

BEGIN (* function arcsin *)
   IF X = 0.0 THEN ARCSIN := 0.0
   ELSE
      IF X = 1.0 THEN ARCSIN := 90.0
      ELSE
         IF X = -1.0 THEN ARCSIN := -90.0
         ELSE ARCSIN := ATAN( 1.0, X/SQRT(1.0 - SQR(X)))
END (* function arcsin *);
```

Figura 1.10 — La funzione Arcoseno.

```
FUNCTION ARCCOS(X : REAL): REAL;
(* arccos in gradi *)
(* è richiesta la funzione ATAN *)
(* 1 Feb 81 *)

BEGIN (* function arccos *)
   IF X = 0.0 THEN  ARCCOS := 90.0
   ELSE
      IF X = 1.0 THEN ARCCOS := 0.0
      ELSE
         IF X = -1.0 THEN ARCCOS := 180.0
         ELSE ARCCOS := ATAN( X/SQRT(1.0 - SQR(X)),1.0)
END (* function arccos *);
```

Figura 1.11 — La funzione Arco-coseno.

FILE

La figura 1.12 dà un esempio di utilizzo della direttiva INCLUDE e dell'istruzione EXTERN. Entrambe fanno riferimento a parti di un programma Pascal che si trovano su un file su un disco. I compilatori Pascal hanno in genere solo una di queste opzioni, e talvolta neanche una. L'uso di queste opzioni presenta diversi vantaggi: prima di tutto il programma principale sarà meno disordinato e più facile da capire se alcune procedure o funzioni sono trattate separatamente; inoltre diversi programmi principali possono richiamare le stesse procedure su file esterni, riducendo in questo modo lo spazio necessario su disco, non essendo necessaria una copia separata della procedura per ogni programma. Altro vantaggio è che un file esterno può essere modificato con più facilità, essendocene una sola copia.

```
PROGRAM TASIN(INPUT, OUTPUT);
(* test arcsin ed arctan *)

VAR
   X : REAL;

(* FUNCTION atan(x, y : real): real;
extern; *)
(*$I ATAN.PAS *)

(* FUNCTION arcsin(x : real): real;
extern; *)
(*$I ASIN.PAS *)

BEGIN (* programma principale *)
   REPEAT
      WRITE(' X:');
      READLN(X);
      WRITELN('L' Arcoseno di' ,X:5:2, 'è' ARCSIN(X):7:4)
   UNTIL X = -1.0
END.
```

Figura 1.12 — Programma di prova delle funzioni Arcoseno e Arcotangente.

La direttiva INCLUDE

Il comando INCLUDE è una direttiva per il compilatore della forma:

```
(* $I NAME.PAS*)          (versione UCSD)
```

oppure

```
(* $F NAME.PAS*)          (versíone Pascal/M)
```

Quando il compilatore incontra la direttiva INCLUDE, va al file sorgente indicato nella direttiva ed elabora le istruzioni che vi trova; quando incontra un carattere di end-of-file, torna al programma principale. Il codice oggetto risultante si presenterà come se il file indicato con la INCLUDE fosse effettivamente inserito nel programma principale.

La direttiva INCLUDE è usata come commento, e quindi, dal punto di vista sintattico, viene ignorata dal compilatore; in genere non può essere nidificata, e quindi un file richiamato con una INCLUDE non può contenere una direttiva INCLUDE per un altro file.

Notiamo che i file richiamati cona INCLUDE sono programmi ASCII, e quindi occupano nel disco lo stesso spazio che occuperebbero se facessero fisicamente parte del programma principale; in realtà, lo spazio di disco usato per il programma principale e i file della INCLUDE, può essere anche maggiore, visto che c'è una dimensione minima di blocco per i file su disco. Inoltre la procedura su file deve essere compilata ogni volta che deve essere compilato il programma principale, e quindi non si risparmia tempo. Ciononostante, la direttiva INCLUDE resta vantaggiosa quando una stessa procedura viene usata da più programmi.

L'istruzione EXTERN

L'istruzione **EXTERN**, disponibile su alcuni compilatori Pascal, è analoga alla direttiva INCLUDE, ma molto più sofisticata. In questo caso una procedura può essere compilata separatamente; il file esterno deve contenere le dichiarazioni **CONST, TYPE** e **VAR** all'inizio, in modo che siano definite le matrici usate come parametri. Per esempio:

```
CONST
  MAXR = 7;
TYPE
  ARY2 = ARRAY[1..MAXR, 1..MAXC]OF REAL;
VAR
  A, Y : ARY2;
```

Segue il corpo della procedura:

```
PROCEDURE NAME(A, Y : ARY);
  ...
END.
```

Dopo l'ultima istruzione di **END** viene messo un punto per indicare la fine del file; alcuni compilatori vogliono il punto e virgola prima del punto.

Il programma principale richiama il file esterno con la normale intestazione della procedura o della funzione, seguita dalla parola **EXTERN**. Per esempio:

```
PROCEDURE PLOT(X, Y, YCALC : ARY;
               N : INTEGER);
EXTERN;
```

L'istruzione **EXTERN** non fa parte del Pascal standard, sebbene menzionata da Jensen e Wirth; in genere nei compilatori si trova più raramente della INCLUDE.

Esiste nel Pascal UCSD, Pascal/MT+ e Pascal/Z. In questo libro vengono usate entrambe, e le parti **EXTERN** sono contenute tra caratteri di commento, che devono essere eliminati se si desidera utilizzarle.

Continuiamo ora il nostro studio delle funzioni logaritmiche ed esponenziali del Pascal.

UNA FUNZIONE POTENZA DEL 10

Può capitare di voler elevare a potenza un numero, ad esempio 10 alla x. In BASIC l'espressione:

```
Y = 10 ↑ X
```

definisce la variabile Y come 10 elevato alla X; se X vale 0.1, Y assume il valore 1.2589...L'espressione corrispondente in FORTRAN è:

```
Y = 10**X
```

Purtroppo nel Pascal standard non è prevista una funzione del genere; è prevista però la funzione EXP, tramite la quale può essere calcolato un elevamento a potenza come segue:

```
Y := EXP(X*LOG10)
```

dove LOG10 è una costante definita precedentemente come il logaritmo naturale di 10. Più in generale, si può calcolare Z elevato a X con l'espressione:

```
Y := EXP(X*LN(Z))
```

dove LN è la funzione logaritmo naturale.

Calcolo di EXP con lo sviluppo in serie di Taylor

Se non esiste la funzione EXP non si possono usare le espressioni precedenti. In questo caso si può sviluppare l'elevamento a potenza in serie di Taylor, con la formula:

$$f(x) = f(a) + (x-a)f'(a) + (x-a)^2 f''(a) / 2! \ldots$$

Se la funzione viene sviluppata intorno allo zero, a vale zero, e i termini dello sviluppo sono:

$$\begin{aligned} f(x) &= 10^x & f(0) &= 1 \\ f'(x) &= 10^x \ln(10) & f'(0) &= \ln(10) \\ f''(x) &= 10^x [\ln(10)]^2 & f''(0) &= [\ln(10)]^2 \end{aligned}$$

La serie risultante è:

$$f(x) = 1 + x \ln 10 + x^2 \frac{(\ln 10)^2}{2!} + x^3 \frac{(\ln 10)^3}{3!} + \ldots$$

Lo sviluppo in serie ha una buona approssimazione intorno allo zero, e diventa sempre meno preciso man mano che aumenta il valore dell'argomento. I coefficienti di una serie con 10 termini sono:

$$\begin{aligned} t_0 &= 1.0 \\ t_1 &= 2.3025851 \\ t_2 &= 2.6509491 \\ t_3 &= 2.0346786 \\ t_4 &= 1.1712551 \\ t_5 &= 0.5393829 \\ t_6 &= 0.2069958 \\ t_7 &= 0.0680894 \\ t_8 &= 0.0195977 \\ t_9 &= 0.0050139 \end{aligned}$$

L'equazione corrispondente è:

$$10^x = \sum_{i=0}^{9} t_i x^i$$

La serie raggiunge una precisione di 6 cifre quando l'argomento è compreso tra 0.5 e −0.5, mentre diventa meno precisa quando l'argomento esce da questi limiti.

Si possono usare degli accorgimenti per ricondurre sempre l'argomento entro questi limiti; per esempio, se è maggiore di 0.5, si può sottrarre 1 tante volte quanto è necessario perché diventi minore di 0.5. Naturalmente il risultato deve essere moltiplicato per 10 ad ogni sottrazione. Per esempio, supponiamo che l'argomento sia 3.5. Allora:

$$10^{3.5} = 10^3 \cdot 10^{0.5} = 10 \cdot 10^{2.5} = 100 \cdot 10^{1.5} = 1000 \cdot 10^{0.5}$$

Il valore di 10 elevato a 3.5 si ottiene dal prodotto di 1000 per $10^{0.5}$.

Analogamente, argomenti minori di −0.5 possono essere incrementati fino a diventarne maggiori, dividendo per 10 il risultato ad ogni addizione. Ora studieremo l'implementazione di questa serie di Taylor con un programma Pascal.

PROGRAMMA PASCAL: CALCOLO DI UNA POTENZA DI 10

La funzione POWER della figura 1.13 può essere usata per calcolare le potenze di 10 da un sviluppo in serie di Taylor con 10 termini. In questa funzione viene eseguito un controllo sul valore dell'argomento in diverse fasi: l'argomento iniziale viene esaminato all'inizio della funzione, e, se il valore è molto grande, viene restituito un numero grande in floating-point come valore della funzione; se è molto piccolo viene restituito zero. I limiti effettivi per questi due test devono essere stabiliti su misura per il vostro compilatore, usando i risultati del programma della figura 1.1. Per esempio, se capita un underflow subito dopo un valore di 3.333E-18, allora il valore di SMALL deve essere 18. Notate che la funzione POWER viene richiamata in modo ricorsivo quando l'argomento è molto grande.

Se il valore dell'argomento è compreso tra 0.5 e LARGE, si applica una tecnica di riduzione, sottraendo ripetutamente 4, finché l'argomento diventa minore di 4, e poi 1 finchè diventa minore di 0.5. Argomenti negativi vengono trattati analogamente. Quando l'argomento è compreso tra −0.5 e 0.5, si utilizza lo sviluppo in serie di Taylor, ottenendo un risultato preciso fino alla sesta cifra.

Notate che l'argomento iniziale, fornito dal programma chiamante, non viene mai modificato; viene invece fatto un cambiamento di variabile,

```
Y : = X ;
```

quasi all'inizio della funzione. Questo non è realmente necessario in quanto il parametro è passato come valore e non come riferimento.

Calcolo della serie di Taylor

Esistono diversi metodi per calcolare il valore di una serie di Taylor; il più semplice è quello di inserire la serie nel programma nella sua forma matematica usuale:

```
SUM := 1.0 + T1*Y + T2*Y*Y + T3*Y*Y*Y + ...
```

Il vantaggio di questa forma è la chiarezza del codice risultante; è però un metodo assolutamente inefficiente, in quanto il tempo di esecuzione cresce drammaticamente al crescere del numero dei termini della serie.

Un altro metodo prevede l'uso di cicli; viene usato un insieme di istruzioni come quello seguente

```
SUM : = 1.0;
TERM : = 1.0;
FOR I : = 1 TO 9 DO
  BEGIN
    TERM : = TERM*Y*LOG10/I;
    SUM : = SUM + TERM
    END;
```

che aggiunge un termine alla serie ogni volta che viene eseguito il ciclo; questo metodo è molto più facile da implementare di quello scelto nel nostro esempio, e la lunghezza della serie può essere rapidamente modificata per adattarlo a diversi package floating-point; è però piuttosto lento.

Il metodo usato nel nostro programma consiste nella somma diretta di un numero fisso di termini polinomiali, i cui coefficienti sono definiti come costanti all'inizio della funzione; la somma viene effettuata con un ciclo sui termini del polinomio. È il metodo più veloce dei tre; ha lo svantaggio di dover incorporare più di 50 cifre nella funzione.

L'uscita del programma è mostrata nella figura 1.14.

```
PROGRAM TPOW( OUTPUT);
(* test sulla funzione potenza-di-dieci *)

CONST
   LOG10 = 2.3025851;

VAR
   I : INTEGER;
   X, Y, A : REAL;

FUNCTION POWER(X : REAL) : REAL;
(* calcola dieci elevato ad x *)
(* per mezzo di un polinomio di decimo grado *)
(* 5 Feb 81 *)

CONST
   BIG = 16.0;
   T1 = 2.3025851;
   T2 = 2.6509491;
   T3 = 2.0346786;
   T4 = 1.1712551;
   T5 = 0.5393829;
   T6 = 0.2069958;
   T7 = 0.0680894;
   T8 = 0.0195977;
   T9 = 0.0050139;

VAR
   SUM, Y, ANS : REAL;
```

Figura 1.13 — Una funzione per calcolare le potenze del 10.

```
BEGIN (* function power *)
  IF X > BIG THEN POWER := POWER(BIG)
  ELSE
    IF X < -BIG THEN POWER := 0.0
    ELSE
      BEGIN
        Y := X;
        ANS := 1;
        WHILE Y > 4.0 DO
          BEGIN (* decrementa di 4 *)
            Y := Y - 4.0;
            ANS := 10000.0 * ANS
          END;
        WHILE Y < -4.0 DO
          BEGIN (* incrementa di 4 *)
            Y := Y + 4.0;
            ANS := 0.0001 * ANS
          END;
        WHILE Y > 0.5 DO
          BEGIN (* decrementa di 1 *)
            Y := Y - 1.0;
            ANS := 10.0 * ANS
          END;
        WHILE Y < -0.5 DO
          BEGIN (* incrementa di 1 *)
            Y := Y + 1.0;
            ANS := 0.1 * ANS
          END;
        SUM := (T5 + Y*(T6 + Y*(T7 + Y*(T8 + Y*T9))));
        SUM := 1.0 + Y*(T1 + Y*(T2 + Y*(T3 + Y*(T4 + Y*SUM))));
        POWER := ANS * SUM
      END (* else *)
END (* function power *);
```

Figura 1.13 — Una funzione per calcolare le potenze del 10 (segue).

```
BEGIN (* programma principale *)
  WRITELN;
  FOR I := 1 TO 4 DO (* passo piccolo *)
    BEGIN
      X := 0.2 * I - 0.1;
      A := POWER( X );
      WRITELN(' x=', X:5:2, ', 10^x=', A,
        ', actual=', EXP( X * LOG10));
      Y := -X; (* cambia segno *)
      A := POWER( Y);
      WRITELN(' x=', Y:5:2, ', 10^x=', A,
        ', reale =', EXP( Y * LOG10))
    END;
  FOR I := 1 TO 6 DO (* passo grande *)
    BEGIN
      X := 2.0 * I - 0.5;
      A := POWER( X );
      WRITELN(' x=', X:5:2, ', 10^x=', A,
        ', reale =', EXP( X * LOG10));
      Y := -X; (* cambia segno *)
      A := POWER( Y);
      WRITELN(' x=', Y:5:2, ', 10^x=', A,
        ', reale =', EXP( Y * LOG10))
    END
END.
```

Figura 1.13 — Una funzione per calcolare le potenze del 10 (segue).

```
x= 0.10, 10^x= 1.25893E+00,   reale = 1.25893E+00
x=-0.10, 10^x= 7.94328E-01,   reale = 7.94328E-01
x= 0.30, 10^x= 1.99526E+00,   reale = 1.99526E+00
x=-0.30, 10^x= 5.01187E-01,   reale = 5.01187E-01
x= 0.50, 10^x= 3.16228E+00,   reale = 3.16228E+00
x=-0.50, 10^x= 3.16227E-01,   reale = 3.16228E-01
x= 0.70, 10^x= 5.01187E+00,   reale = 5.01187E+00
x=-0.70, 10^x= 1.99526E-01,   reale = 1.99526E-01
x= 1.50, 10^x= 3.16228E+01,   reale = 3.16228E+01
x=-1.50, 10^x= 3.16227E-02,   reale = 3.16228E-02
x= 3.50, 10^x= 3.16228E+03,   reale = 3.16228E+03
x=-3.50, 10^x= 3.16227E-04,   reale = 3.16228E-04
x= 5.50, 10^x= 3.16228E+05,   reale = 3.16228E+05
x=-5.50, 10^x= 3.16227E-06,   reale = 3.16228E-06
x= 7.50, 10^x= 3.16228E+07,   reale = 3.16228E+07
x=-7.50, 10^x= 3.16227E-08,   reale = 3.16228E-08
x= 9.50, 10^x= 3.16228E+09,   reale = 3.16228E+09
x=-9.50, 10^x= 3.16227E-10,   reale = 3.16228E-10
x= 11.50, 10^x= 3.16228E+11,   reale = 3.16228E+11
x=-11.50, 10^x= 3.16227E-12,   reale = 3.16228E-12
```

Figura 1.14 — Risultati del programma di calcolo delle potenze del 10.

SOMMARIO

In questo capitolo sono stati forniti alcuni strumenti di valutazione dei compilatori Pascal; siamo partiti da un programma di test delle operazioni in floating-point, eseguito sotto tre diversi compilatori con analisi comparata dei risultati. Abbiamo poi scoperto un'interessante «debolezza» delle funzioni SIN e COS di alcuni compilatori e abbiamo scritto un programma per provarle.

Studiando altri programmi di prova di funzioni, ci siamo accorti che ci serviva una tecnica per inserire file esterni in un programma principale; questo ci ha portato a studiare la direttiva INCLUDE e l'istruzione EXTERN del Pascal.

Infine abbiamo usato le conoscenze acquisite in questo capitolo per scrivere un programma Pascal che calcola le potenze del 10 con lo sviluppo in serie di Taylor.

CAPITOLO 2

MEDIA E DEVIAZIONE STANDARD

INTRODUZIONE

In questo capitolo richiameremo alcune nozioni di statistica e studieremo come applicarle con dei programmi Pascal. Descriveremo l'uso della media e della deviazione standard, e presenteremo un programma che calcola questi valori. Discuteremo anche la generazione di numeri casuali con il calcolatore, in particolare in Pascal. Infine studieremo dei programmi Pascal per valutare la "casualità" dei numeri generati.

LA MEDIA

Spesso si usa un solo numero, che si chiama "media" o "valor medio", per rappresentare un insieme di dati. La media si calcola sommando tutti gli elementi dell'insieme e dividendo il risultato per il numero degli elementi, con la formula seguente:

$$\bar{y} = \frac{\sum_{i=1}^{N} y_i}{N} \qquad (1)$$

dove Y_i è l'i-esimo elemento di un insieme che ne contiene N. Il simbolo Y (y segnato) è la media risultante.

Quando la distribuzione dei dati è continua, la media può essere calcolata con un integrale. Consideriamo la funzione y = f(x) nell'intervallo (a,b); la media di y è costante in questo intervallo e quindi l'area della superficie sottesa dalla media è uguale a quella della superficie sottesa dalla curva f(x).

$$\bar{y}\,(b - a) = \int_a^b f(x)dx$$

Perciò il valor medio di y è:

$$\bar{y} = \frac{\int_a^b f(x)dx}{b - a}$$

Dispersione dalla media

Talvolta tutti i dati sono vicini al valor medio, talvolta, invece, i valori coprono una gamma molto ampia. Consideriamo, come esempio del secondo caso, i dati sul tempo: la quantità media di pioggia caduta in un anno a San Francisco è di 19 pollici, e quella di neve a New York City è di 30 pollici, ma alcuni anni sono molto piovosi, e altri molto asciutti; la temperatura media in un anno a San Francisco e ad Albuquerque è esattamente la stessa, 57°, ma il clima di queste due città è completamente diverso.

Consideriamo un altro esempio: le scatole di una certa marca di cereali per colazione presentano la dicitura:

Peso netto 500 gr.

Supponiamo che un ispettore dell'Ufficio Pesi e Misure decida di controllare le scatole di questa marca in un certo negozio: apre diverse scatole e ne pesa il contenuto; alcune scatole contengono esattamente 500 gr., altre 470 gr. o 530 gr.

Le scatole che contengono solo 470 gr. dovrebbero essere confiscate come esempio di peso scarso? Se pesiamo il contenuto di 100 scatole la distribuzione di frequenza risultante assomiglia alla curva della Figura 2.1.

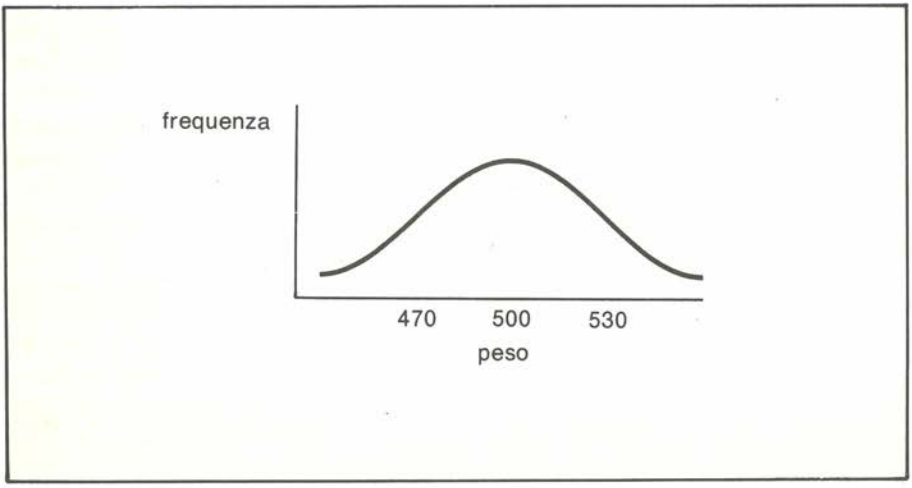


Figura 2.1 — Una curva a forma di Gaussiana.

Il peso medio è 500 gr.; alcune scatole pesano di meno, altre di più; solo pochissime pesano più di 530 gr. o meno di 470 gr.

La curva della figura 2.1 è a forma di campana e mostra quella che si chiama distribuzione "Gaussiana" o "normale". Questo comportamento è tipico di una distribuzione casuale dei dati intorno al valor medio, e l'equazione di questa curva si ricava dalla funzione Gamma e dalla Gaussiana, di cui ci occuperemo nel Capitolo 11.

Ora supponiamo che vengano controllate anche le scatole di un'altra marca. Il risultato si presenta come nella figura 2.2: i dati hanno ancora una distribuzione a forma di campana, con un valor medio di 500 gr., ma questa volta c'è una maggiore dispersione nei pesi; alcune scatole pesano 800 gr., altre 200 gr.

Evidentemente i due casi sono molto diversi, anche se entrambi hanno lo stesso valor medio; dunque ci vuole qualcols'altro per descrivere correttamente la situazione, qualcosa che informi sulla "dispersione". Nel prossimo paragrafo descriveremo lo strumento che possiamo usare, la "deviazione standard".

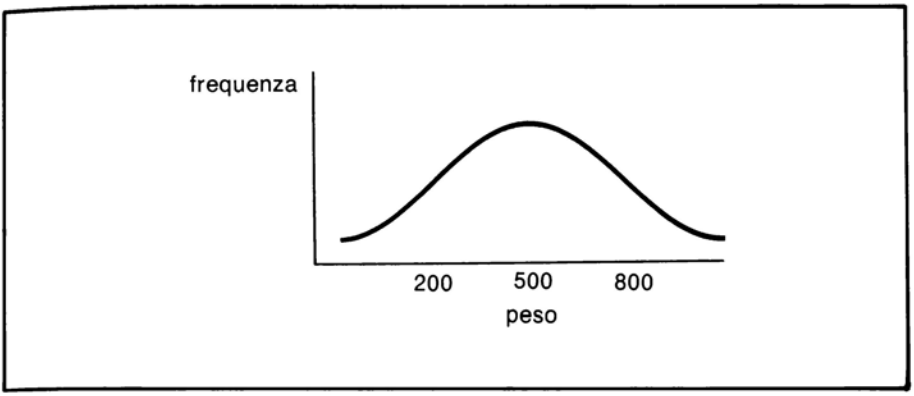


Figura 2.2 — Una maggiore dispersione.

LA DEVIAZIONE STANDARD

La deviazione standard è una misura della dispersione intorno al valor medio: una grande deviazione standard significa una grande dispersione; una piccola deviazione standard significa una piccola dispersione. Il simbolo della deviazione standard è la lettera minuscola "sigma" dell'alfabeto greco. La deviazione standard è definita dalla seguente relazione:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (\bar{y} - y_i)^2}{N - 1}} \qquad (2)$$

dove $\bar{y}$ è la media e gli y_i sono gli N dati.

L'equazione (2) spiega il significato della deviazione standard: in una distribuzione di valori, è importante il quadrato della differenza tra ogni elemento e il valor medio. Se gli elementi sono vicini al valor medio, questa differenza sarà piccola, e anche la corrispondente deviazione standard sarà piccola. Se invece i dati sono distribuiti lontano dal valor medio, la differenza sarà grande, come anche la deviazione standard.

La deviazione standard può essere usata per descrivere quantitativamente la dispersione di un insieme di dati. Ad esempio, in un intervallo pari a 1 deviazione standard intorno alla media è compreso il 68% dei dati; il 95% dei dati è invece compreso in un intervallo pari a 2 deviazioni standard, e il 99.7% in un intervallo pari a 3 deviazioni standard.

$\bar{y} \pm \sigma$	68%
$\bar{y} \pm 2\sigma$	95%
$\bar{y} \pm 3\sigma$	99.7%

Supponiamo di dover scegliere un tipo di acciaio per costruire un ponte, e di eseguire delle prove per stabilirne la resistenza. Un tipo di acciaio presenta una resistenza di 450 MPa (megapascal) con una deviazione standard di 10 MPa. I risultati delle prove ci dicono che, statisticamente, il 99.7% dei pezzi hanno una resistenza compresa tra 420 e 480 MPa, cioè entro 3 deviazioni standard, e quindi il progetto può contare su di una resistenza di 420 MPa, 3 deviazioni standard al di sotto del valor medio.

Un secondo tipo di acciaio ha una resistenza media di 500 MPa. È migliore del primo? Che cosa possiamo dire se la deviazione standard di questo secondo tipo è 40 MPa? Il valore di questa deviazione standard significa una maggiore dispersione; una resistenza di 3 deviazioni standard al di sotto della media significa, per questo secondo tipo di acciaio, solo 380 MPa; non è quindi conveniente, anche se il valor medio è più alto.

I metalli usati per le costruzioni devono avere in genere piccole deviazioni standard. Materiali fragili, come il cemento armato e il vetro hanno invece, in genere, grandi deviazioni standard; supponiamo di fare delle prove per un tipo di vetro da usare per una porta principale di un ufficio. La resistenza media è di 120 MPa, e la deviazione standard è di 40 MPa; 3 deviazioni standard al di sotto della media significa 0; è quindi molto importante considerare la deviazione standard altrettanto seriamente che il valor medio.

Abbiamo imparato il significato e l'importanza della deviazione standard; ora impareremo a calcolare il suo valore con un programma in Pascal.

Calcolo della deviazione standard

L'equazione 2 è una corretta rappresentazione della deviazione standard, ma non dà un buon metodo di calcolo; infatti la sottrazione di ogni elemento dal valor medio

provoca una perdita di significatività, soprattutto per i punti molto vicini alla media. Più piccola è la deviazione standard, più grosso diventa il problema. Un altro svantaggio è che la media deve essere calcolata prima di effettuare qualsiasi sottrazione, mentre questo calcolo non è possibile finchè non sono disponibili tutti i dati.

Sviluppando il numeratore dell'equazione 2 si trova un buon metodo di calcolo della deviazione standard.

$$\Sigma\,(\bar{y}^2 - 2\bar{y}y_i + y_i^2)$$

Combinando l'equazione precedente con l'equazione 1, abbiamo:

$$\left(\frac{\Sigma y_i}{N}\right)^2 N - 2\left(\frac{\Sigma y_i}{N}\right)\Sigma y_i + \Sigma y_i^{\,2}$$

dato che $\bar{y} = \Sigma y_i / N$. La formula risultante è:

$$\sigma = \sqrt{\frac{\Sigma y^2 - \Sigma y\,\Sigma y/N}{N-1}}$$

Con questo metodo si conservano due totali: quello dei singoli dati e quello dei loro quadrati.

PROGRAMMA PASCAL: MEDIA E DEVIAZIONE STANDARD

Il programma della figura 2.3 può essere usato per calcolare la media e la deviazione standard di un insieme di numeri. Il programma principale chiede all'utente il numero di elementi, e chiama la procedura MEANSTD per calcolare la media e la deviazione standard. Questa procedura usa quattro parametri: il primo è la matrice che contiene i valori, dichiarata di tipo ARY; il secondo è il numero di valori della matrice; il terzo e il quarto sono la media e la deviazione standard, calcolati dalla procedura stessa.

Il programma della figura 2.3 mostra come si possono passare matrici come parametri alle procedure. Il Pascal richiede un'istruzione di dichiarazione particolare per ogni parametro; un'espressione del tipo

```
PROCEDURE MEANSTD(X:ARRAY[1..MAX]OF REAL;...);
```

non è consentita, mentre nel programma principale deve essere definito esplicitamente il tipo di ogni matrice usata come parametro.

```
TYPE
  ARY = ARRAY[1..MAX]OF REAL
```

```
PROGRAM MEANS(INPUT, OUTPUT);
(* Stima della media e della deviazione standard *)
(* 19 Gen 81 *)

CONST
   MAX = 80;

TYPE
   ARY = ARRAY[1..MAX] OF REAL;

VAR
   X          : ARY;
   I,N        : INTEGER;
   MEAN, STD  : REAL;

PROCEDURE MEANSTD
                    ( X : ARY; (* matrice dei valori *)
      LENGTH          : INTEGER;
      VAR MEAN        : REAL;
      VAR STD_DEV     : REAL);

VAR
   I                  : INTEGER;
   SUM_X, SUM_SQ: REAL;

BEGIN
   SUM_X := 0;
   SUM_SQ := 0;
   FOR I := 1 TO LENGTH DO
      BEGIN
         SUM_X := SUM_X + X[I];
         SUM_SQ := SUM_SQ + X[I] * X[I]
```

Figura 2.3 — Calcolo della media e della deviazione standard.

```
      END;
   MEAN := SUM_X/LENGTH;
   STD_DEV :=
         SQRT((SUM_SQ - SQR( SUM_X)/LENGTH)/(LENGTH-1))
END (* procedure meanstd *);

BEGIN (* programma principale *)
   WRITELN;
   WRITELN ('Calcolo della media e della deviazione standard');
   REPEAT
      WRITE ('Quanti punti?');
      READLN(N)
   UNTIL N <= MAX;
   FOR I:=1 TO N DO
      BEGIN
         WRITE( I:3,': ');
         READLN(X[I])
      END;
   MEANSTD( X, N, MEAN, STD);
   WRITELN;
   WRITELN ('Per', N:3, 'punti, media=', MEAN:8:4,
            ', sigma=', STD:8:4)
END.
```

Figura 2.3 — Calcolo della media e della deviazione standard (segue).

Il tipo così definito è poi inserito nell'intestazione della procedura.

```
PROCEDURE MEANSTD(X: ARY;...);
```

Il programma inizia chiedendo all'utente il numero di valori da leggere; se è maggiore della dimensione della matrice X, la domanda viene ripetuta. Una volta letti i valori, vengono calcolate la media e la deviazione standard.

Se devono essere inseriti molti valori, si può usare un altro metodo: il programma conta i numeri mano a mano che vengono letti, e un valore particolare, ad esempio un numero minore di −20000, viene usato per segnalare la fine dei dati.

Compilate il programma ed eseguitelo. Inserite i cinque numeri 1,2,3,4,5, e verificate che la media è 3 e la deviazione standard 1.58. I risultati si presenteranno come in figura 2.4.

```
Calcolo della media e della deviazione standard
Quanti punti? 5
 1:  1
 2:  2
 3:  3
 4:  4
 5:  5

Per 5 punti, media=   3.0000, sigma=   1.5811
```

Figura 2.4 — Risultati del programma di calcolo della media e della deviazione standard.

Abbiamo così completato l'introduzione alla media e alla deviazione standard; vi ritorneremo in seguito. Ora affronteremo il problema dei numeri casuali e della loro generazione.

NUMERI CASUALI

Quando non sono disponibili dati reali per provare un programma, può essere necessario usare, in certi casi, dei numeri casuali. Supponiamo per esempio di aver scritto un programma per interpolare con una retta un insieme di dati sperimentali: dovremo generare un'insieme di punti distribuiti su una linea retta, e poi allontanarli da essa usando una funzione di generazione di numeri casuali.

FUNZIONE PASCAL: GENERAZIONE DI NUMERI CASUALI

In alcuni, ma non tutti, compilatori Pascal esiste la funzione di generazione di numeri casuali; per alcuni programmi di questo libro questa funzione è necessaria. Se il vostro compilatore non la possiede, potete usare il programma della figura 2.5, che è stato studiato in modo da essere compatibile con le più comuni funzioni di generazione di numeri casuali fornite normalmente dai compilatori Pascal.

Questo programma restituisce una sequenza di numeri compresi tra zero e uno. Una chiamata tipica alla funzione RANDOM sarà del tipo:

```
XI : = RANDOM(0);
```

Alcuni generatori di numeri casuali usano il parametro come origine per il numero successivo; in altri casi, un particolare valore dell'argomento segnala che verrà generato nuovamente il valore appena generato. Ma in questo caso il parametro per la funzione RANDOM è un argomento "dummy", e può quindi essere usato qualsiasi numero intero.

L'origine effettiva per la generazione dei numeri casuali è contenuta nella variabile reale globale SEED, che deve essere dichiarata ed inizializzata dal programma chiamante. Una buona scelta può essere il valore iniziale di 4.0. La procedura RANDOM trasgredisce una regola generale della programmazione, in base alla quale una subroutine non dovrebbe cambiare variabili globali; ma in questo caso è probabilmente la soluzione migliore; infatti il Pascal non consente variabili locali permanenti nelle subroutine.

Dopo la prima chiamata alla funzione RANDOM, SEED contiene il numero casuale appena generato, ma questo numero non può essere memorizzato come variabile locale nella RANDOM. Infatti, in Pascal, le variabili locali sono sempre reinizializzate ad ogni esecuzione di una procedura. Un'altra possibilità è di dichiarare l'origine come parametro della funzione RANDOM; in questo caso il programma chiamante deve conservare una copia dell'ultimo numero casuale da usare come origine per la chiamata successiva. Questo però va contro lo stile delle funzioni di generazione di numeri casuali che si trovano di solito nei compilatori Pascal; si è scelto quindi di memorizzare l'origine come variabile globale.

Studieremo ora un programma di prova delle funzioni di generazione di numeri casuali.

```
FUNCTION RANDOM(DUMMY: INTEGER): REAL;
(* numero random 0 — 1 *)
(* definisce SEED = 4.0 come globale *)
(* Adattato dai Programmi Applicativi HP — 35 *)

CONST
   PI = 3.14159;
VAR
   X: REAL;
   I: INTEGER;

BEGIN (* Random *)
   X := SEED + PI;
   X := EXP(5.0 * LN(X));
   SEED := X — TRUNC(X);
   RANDOM := SEED
END(* Random *);
```

Figura 2.5 — Una funzione Pascal per generare numeri casuali.

PROGRAMMA PASCAL: VALUTAZIONE DELLE FUNZIONI DI GENERAZIONE DI NUMERI CASUALI

Sia che usiate il programma della figura 2.5 che una routine fornita dal vostro Pascal, potete eseguire un semplice test per valutarne la bontà. Il valor medio di un insieme di numeri casuali compresi tra 0 e 1 è ovviamente 1/2, e la deviazione standard è uguale al reciproco della radice quadrata di 12, cioè 0.2887.

Il programma della figura 2.7 è un test per le funzioni di generazione dei numeri casuali. Chiama la funzione RANDOM 48 volte, poi chiama la procedura MEANSTD della figura 2.3, per calcolare la media e la deviazione standard. Questo processo viene ripetuto 20 volte, ed ogni volta vengono stampate la media e la deviazione standard. I valori previsti sono stampati in cima alle colonne dei risultati.

Esecuzione del programma

Scrivete il programma, aggiungete la procedura MEANSTD e, se necessario, la funzione RANDOM; poi eseguite la prova. Le funzioni di generazione di numeri casuali sono spesso dette "pseudo-casuali", per sottolineare il fatto che i numeri generati non sono perfettamente casuali; perciò non dovete aspettarvi di ottenere un va-

```
media      dev std
(0.5)      (0.2887)
====================
0.4294     0.3156
0.6200     0.3016
0.5008     0.2439
0.3615     0.2758
0.4121     0.2522
0.3960     0.2554
0.5444     0.2363
0.5013     0.3129
0.5282     0.3734
0.5105     0.2375
0.5585     0.2701
```

Figura 2.6 — Risultati del programma RANTST.

lor medio pari a 0.5 e una deviazione standard di 0.2887 per ogni insieme di 48 valori. La prima parte dei risultati prodotti dal programma RANTST si presenteranno come in figura 2.6.

Notate che il valor medio varia da 0.36 a 0.62, e la deviazione standard da 0.24 a 0.37. In genere le funzioni fornite dai compilatori Pascal sono più precise, ma non sempre.

Generazione di numeri casuali con π

Una breve sequenza di numeri casuali si può ottenere dalle prime 15 cifre di π:

3.14159265358979

con un valor medio di 5.1 e una deviazione standard di 2.8.

La funzione di generazione di numeri casuali che descriveremo ora è studiata per simulare dati sperimentali.

Distribuzione Gaussiana di numeri casuali

Gli errori che si commettono effettuando una misura possono essere di diverse entità, ma quelli piccoli sono più probabili di quelli grandi. Per esempio, supponiamo di misurare con un metro una tavola lunga due metri; un errore di un metro è certamente meno probabile di un errore di un centimetro; un errore di cinque metri è ancora più improbabile. In genere, valori molto lontani dal valore esatto sono molto meno probabili di quelli vicini; perciò, se vogliamo simulare dati sperimentali con dei numeri casuali, questi non possono avere una distribuzione uniforme, mentre le più comuni funzioni generano numeri casuali uniformemente distribuiti tra zero e uno.

Per simulare dati sperimentali occorre un insieme di numeri casuali raggruppati intorno al valor medio, la cui distribuzione di frequenza avrà una forma a campana, come la distribuzione normale o Gaussiana. Fortunatamente è abbastanza facile produrre una simile distribuzione da una comune funzione di generazione di numeri casuali.

Consideriamo una sequenza di 12 numeri casuali; è altamente improbabile che tutti e 12 siano 0, come anche che siano 1. Supponiamo ora di generare un nuovo numero dalla somma di 12 numeri casuali distribuiti uniformemente; il valor medio dei numeri di partenza è 1/2, la somma sarà circa 12 volte il valor medio, cioè 6. Come vedremo questa è la chiave per generare numeri casuali secondo una distribuzione Gaussiana, usando una normale funzione di generazione di numeri casuali; studieremo un'implementazione di questa funzione in Pascal.

```
PROGRAM RANTST(OUTPUT);
(* test sul generatore di numeri random RANDOM *)
(* sono inoltre richieste la procedura MEANSTD e la
        funzione RANDOM *)

TYPE
   ARY = ARRAY[1..100] OF REAL;

VAR
   X          : ARY;
   N, I, J    : INTEGER;
   R, MEAN, STD,
      SEED    : REAL;

(* FUNCTION random(dummy: integer): real;
extern; *)

(*$F RANDOM.PAS *)

(* PROCEDURE meanstd
                    ( x  : ary;
          length         : integer;
      VAR mean           : real;
      VAR std_dev        : real);
extern; *)

(*$F MEANSTD.PAS *)
```

Figura 2.7 — Programma di prova della funzione di generazione dei numeri casuali.

```
BEGIN  (* Programma principale *)
   SEED := 4.0;
   N := 48;
   WRITELN;
   WRITELN( '  media   dev std');
   WRITELN( '   (0.5)    (0.2887)' );
   WRITELN( '  ====================');
   FOR J := 1 TO 20 DO
      BEGIN
         FOR I := 1 TO N DO
            X[I] := RANDOM(0);
         MEANSTD( X, N, MEAN, STD);
         WRITELN( MEAN :10:4, STD :10:4)
      END (* ciclo j *)
END.
```

Figura 2.7 — Programma di prova della funzione di generazione dei numeri casuali (segue).

FUNZIONE PASCAL: DISTRIBUZIONE GAUSSIANA DI NUMERI CASUALI

La figura 2.8 mostra una funzione Pascal che genera numeri casuali secondo una distribuzione Gaussiana. Ogni numero è ottenuto sommando 12 numeri casuali distribuiti uniformemente, e sottraendo 6. Un insieme di numeri così generati dovrebbe aver un valor medio 0 e una deviazione standard 1. Possono essere scelti altri valori. La formula per calcolare ogni numero casuale, RANDG, è:

RANDG : = SIGMA* (SUM - 6) + MEAN

dove SUM è la somma di 12 numeri casuali uniformemente distribuiti, SIGMA è la deviazione standard desiderata, e MEAN il valor medio. Se ogni numero è ottenuto da più di 12 numeri, la formula diventa:

```
RANDG : = SIGMA* (SUM - NUM/2)* SQRT(12/NUM) + MEAN
```

dove NUM è la quantità di numeri casuali uniformemente distribuiti, usati per generare la distribuzione Gaussiana.

Dunque la funzione RANDG può essere usata con la RANDOM per generare un insieme di numeri casuali con distribuzione Gaussiana. Il programma chiamante deve fornire due parametri: la media e la deviazione standard volute. Oppure si possono assegnare due valori costanti a MEAN e SIGMA nella funzione RANDG, senza quindi farli comparire come parametri.

```
FUNCTION RANDG(MEAN, SIGMA : REAL): REAL;
(* produzione di numeri random con una distribuzione Gaussiana *)
(* MEAN e SIGMA sono fornite per mezzo di una chiamata
   di programma *)
(* è richiesta la funzione RANDOM *)

VAR
   I      : INTEGER;
   SUM    : REAL;

BEGIN
   SUM := 0.0;
   FOR I  := 1 TO 12 DO
      SUM := SUM + RANDOM(0);
   RANDG := (SUM - 6.0) * SIGMA + MEAN
END (* function randg *);
```

Figura 2.8 — Una funzione di generazione di numeri casuali secondo la distribuzione normale.

PROGRAMMA PASCAL: VALUTAZIONE DELLA FUNZIONE RANDG

Nella figura 2.9 c'è un programma di prova della funzione RANDG. La media e la deviazione standard desiderate sono rispettivamente 10 e 1/2, e la procedura MEANSTD viene chiamata per calcolare la media e la deviazione standard di ogni insieme di numeri casuali; i numeri generati dovranno avere una media di 10 e una deviazione standard di 1/2.

L'area della superficie sottesa dalla curva di distribuzione normale dipende dalla deviazione standard, e può essere ottenuta dalla funzione di errore Gaussiana presentata nel Capitolo 11.

```
PROGRAM RANTST(OUTPUT);
(* test sul generatore di numeri random Gaussiani RANDG *)
(* è inoltre richiesta la procedura MEANST
   e la funzione RANDOM *)

TYPE
  ARY = ARRAY[1..100] OF REAL;

VAR
  X          : ARY;
  N, I, J    : INTEGER;
  R, AVER, STD,
     SEED    : REAL;

(* FUNCTION random(dummy: integer): real;
extern; *)
(*$F RANDOM.PAS *)

(* PROCEDURE meanstd
                    ( x  : ary;
        length          : integer;
```

Figura 2.9 — Programma di prova della funzione Gaussiana di generazione di numeri casuali.

```
        VAR mean       : real;
        VAR std_dev    : real);
extern; *)
(*$F MEANSTD.PAS *)

(* FUNCTION randg(mean, sigma : real): real;
extern; *)
(*$F RANDG.PAS *)

BEGIN  (* Programma principale *)
   SEED := 4.0;
   N := 50;
   WRITELN;
   WRITELN( '  media dev std' );
   WRITELN( '    (10)       (0.5)' );
   WRITELN( '  ====================');
   FOR J := 1 TO 20 DO
      BEGIN
         FOR I := 1 TO N DO
            X[I] := RANDG(10, 0.5);
         MEANSTD( X, N, AVER, STD);
         WRITELN( AVER :10:4, STD :10:4)
      END (* ciclo j *)
END.
```

Figura 2.9 — Programma di prova della funzione Gaussiana di generazione di numeri casuali (segue).

SOMMARIO

Abbiamo iniziato questo capitolo con la discussione di due importanti strumenti di statistica, la media e la deviazione standard, giungendo alla stesura di un programma in Pascal per il calcolo di questi due valori. Abbiamo poi scoperto come scrivere due diverse funzioni Pascal di generazione di numeri casuali, che potranno essere utili se il nostro compilatore non possiede tali funzioni. Infine abbiamo dimostrato l'importanza di poter valutare la "bontà" dei numeri casuali generati, e abbiamo studiato due programmi che lo fanno.

Nel prossimo capitolo approfondiremo la nostra conoscenza di Pascal, studiando come eprimere vettori e matrici sotto forma di matrici Pascal.

CAPITOLO 3

OPERAZIONI SU VETTORI E MATRICI

INTRODUZIONE

In quasi tutti i capitoli di questo libro sono presentati programmi che fanno uso di vettori e di matrici; in questo capitolo richiameremo dunque i concetti di vettore e matrice e alcune tra le più comuni operazioni matematiche su di essi. Studieremo poi alcuni programmi Pascal sull'argomento. Ricordiamo per inciso che i vettori e le matrici sono un utilissimo strumento, che semplifica notevolmente la programmazione di operazioni matematiche su insiemi di dati.

SCALARI E MATRICI

Cominciamo a stabilire la differenza tra una variabile scalare e una matrice. Una variabile semplice si chiama scalare, è identificata da un unico nome simbolico, ed è associata ad un solo valore. Per esempio, l'espressione Pascal

```
YEAR : = 1971
```

assegna un valore alla variabile scalare YEAR. In Pascal le variabili scalari possono essere reali, intere, alfanumeriche o logiche(booleane).

Talvolta è necessario fare riferimento ad un insieme di variabili; il Pascal prevede diversi tipi di strutture dati per questo scopo, di cui la più semplice è quella che viene chiamata "array". In un array tutti gli elementi sono dello stesso tipo, cioè reali, interi, alfanumerici o logici; inoltre possono essere essi stessi array, ma nei casi più frequenti sono variabili scalari.

Gli elementi di un array sono identificati da un unico nome simbolico, mentre la loro posizione nell'array è identificata da un indice che segue il nome. L'elemento che occupa una certa posizione può essere richiamato individualmente tramite il

suo indice, e il suo valore può essere modificato senza toccare gli altri elementi. Nei prossimi paragrafi vedremo come si rappresentano vettori e matrici sotto forma di array in Pascal, cominciando dai vettori.

VETTORI

Un vettore è un array ad una dimensione. (Il numero delle dimensioni è il numero degli indici, e non degli elementi). Ogni elemento di un vettore è identificato da un indice che, in genere, va da 1 al numero massimo degli elementi del vettore; in Pascal l'indice può partire da qualsiasi valore, cioè un array può iniziare con indice 0 o 101.

Nell'uso comune, gli elementi di un vettore sono separati da uno spazio. Consideriamo ad esempio il vettore v che contiene i valori:

2 5 1 9 4 3

L'istruzione Pascal:

```
VAR V : ARRAY[1..6]OF INTEGER;
```

definisce il simbolo V come vettore contenente interi. Il numero massimo di elementi(la lunghezza) è 6. Agli elementi del vettore possono essere assegnati dei valori con istruzioni Pascal del tipo:

```
V[1] := 2;
V[2] := 5;
V[3] := 1;
V[4] := 9;
V[5] := 4;
V[6] := 3;
```

Il primo elemento del vettore si trova alla posizione V[1], il secondo alla posizione V[2] e l'ultimo a V[6]. Il primo elemento ha valore 2, il secondo 5 e l'ultimo 3.

Dal momento che i vettori hanno una sola dimensione, non ha importanza se sono scritti orizzontalmente o verticalmente; talvolta però è necessario distinguere tra vettori riga, scritti orizzontalmente, e vettori colonna, scritti verticalmente. In questo caso, l'insieme:

$$\begin{bmatrix} 2 \\ 5 \\ 1 \\ 9 \\ 4 \\ 3 \end{bmatrix}$$

è un vettore colonna.

Nel prossimo paragrafo studieremo le operazioni sui vettori e la loro implementazione in Pascal.

Aritmetica dei vettori

Le più comuni operazioni sui vettori sono: modulo moltiplicazione di uno scalare per un vettore, somma tra vettori, prodotto scalare, prodotto vettoriale. Consideriamo ora queste operazioni una alla volta.

Modulo (o ampiezza)

L'ampiezza di un vettore è un valore scalare, che si ottiene estraendo la radice quadrata della somma dei quadrati degli elementi del vettore; per esempio, l'ampiezza del vettore Y:

$$\mathbf{Y} = [2\ 2\ 1]$$

è uguale alla radice quadrata di 4 + 4 + 1, cioè 3. Il calcolo dell'ampiezza può essere programmato con l'istruzione Pascal:

```
MAG : = SQRT(SQR(Y[1]) + SQR(Y[2]) + SQR(Y[3]))
```

Moltiplicazione di uno scalare per un vettore

Per moltiplicare un vettore y per uno scalare s, si moltiplica ogni elemento di y per s. Per esempio, v per 2 è uguale a

$$2\mathbf{v} = [4\ 10\ 2\ 18\ 8\ 6]$$

La seguente espressione Pascal genera un vettore V2 di cui ogni elmento è il doppio del corrispondente elemento del vettore V:

```
FOR I : = 1 TO 6 DO V2[I]: = 2.0* V[I]
```

Somma di vettori

Due vettori si possono sommare uno all'altro se hanno lo stesso numero di elementi, o la stessa lunghezza; il risultato è un nuovo vettore di cui ogni elemento è la somma degli elementi corrispondenti nei vettori originali. Se

$$\mathbf{a} = [1\ 2\ 3]$$

e

$$\mathbf{b} = [3\ 4\ 5]$$

allora

$$\mathbf{a} + \mathbf{b} = [4\ 6\ 8]$$

L'espressione corrispondente in Pascal è:

```
FOR I : = 1 TO 3 DO C[I]: = A[I] + B[I]
```

Prodotto scalare

Il prodotto scalare tra due vettori della stessa lunghezza è uno scalare ottenuto sommando i prodotti di ogni elemento di un vettore per il corrispondente elemento dell'altro; il simbolo matematico per l'operazione di prodotto scalare è un punto tra i due operandi.

$$\mathbf{a} \cdot \mathbf{b} = (1)(3) + (2)(4) + (3)(5) = 26$$

Il prodotto scalare è uguale al prodotto tra le ampiezze dei due vettori per il coseno dell'angolo formato dai due:

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}|\ |\mathbf{b}| \cos \theta$$

Questa formula può essere usata per trovare l'angolo formato da due vettori. Per esempio, se l'angolo tra due vettori a e b é 10.7°:

$$\cos\theta = 26/(3.742 \cdot 7.071) = 0.9826$$
$$\text{Arccos } 0.9826 = 10.7°$$

Prodotto vettoriale

Il prodotto vettoriale tra due vettori è un terzo vettore perpendicolare ai due, con ampiezza pari al prodotto delle due ampiezze per il seno dell'angolo compreso tra i due vettori originali; il simbolo dell'operazione è x:

$$|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}|\,|\mathbf{b}| \sin\theta$$

Può essere calcolato con le seguenti istruzioni Pascal:

```
C[1] : = A[2]*B[3] - B[2]*A[3];
C[2] : = - A[1]*B[3] + B[1]*A[3];
C[3] : = A[1]*B[2] - B[1]*A[2];
```

Il prodotto vettoriale tra i due vettori a e b è il vettore [–2 4 –2]; la sua ampiezza è 4.9.

Abbiamo visto come programmare in Pascal le principali operazioni aritmetiche sui vettori. In seguito vedremo come eseguire le stesse operazioni su matrici; prima dobbiamo studiare un particolare tipo di vettore, la stringa.

Stringhe

Le stringhe, insiemi di caratteri alfabetici e numerici (alfanumerici), servono per rappresentare informazioni come nomi o indirizzi. In Pascal si possono definire dei vettori di stringa, che contengono nomi o indirizzi; inoltre la stringa deve essere definita come array di caratteri.

```
TYPE
  STRING = PACKED ARRAY[1..15]OF CHAR;
VAR
  ADDRESS : ARRAY[1..100]OF STRING;
```

La dichiarazione **VAR** serve per definire la variabile ADDRESS come array contenente fino a 100 indirizzi. Nel Pascal standard, ogni stringa di indirizzo deve conte-

nere il numero di caratteri dichiarato (15 in questo caso), aggiungendo eventualmente all'indirizzo effettivo degli spazi. Per esempio:

```
ADDRESS[1]: = '1234 Oak St. ';
ADDRESS[1]: = '2222 First St. ';
```

Se il vostro Pascal prevede il tipo di stringa dinamica, la dichiarazione **TYPE** non è necessaria, e ogni stringa può avere una lunghezza qualsiasi, fino a un massimo prestabilito (in genere 80 caratteri). Per esempio:

```
ADDRESS[1]: = '1234 Oak St.';
ADDRESS[2]: = '2222 First St.';
```

In ogni caso, sulle stringhe si possono eseguire operazioni di ordinamento e di altro tipo:

```
IF ADDRESS[I]<ADDRESS[J] THEN...
```

Passiamo ora agli array a due dimensioni e alla loro rappresentazione in Pascal.

MATRICI

Un array a due dimensioni si chiama matrice; gli elementi di un simile insieme di dati sono sistemati in un rettangolo o in un quadrato, e possono essere considerati come un insieme di righe orizzontali (vettori riga), o di righe verticali (vettori colonna). Una matrice di questo tipo può essere definita come un insieme monodimensionale di vettori.

Ogni elemento della matrice è definito da una coppia di indici: l'indice di riga e l'indice di colonna. Consideriamo, ad esempio, la matrice:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & & x_{1m} \\ x_{21} & x_{22} & x_{23} & & x_{2m} \\ x_{31} & x_{32} & x_{33} & & x_{3m} \\ \cdots & \cdots & \cdots & & \cdots \\ x_{n1} & x_{n2} & x_{n3} & & x_{nm} \end{bmatrix}$$

che contiene n righe e m colonne. Viene sempre indicato prima l'indice di riga e poi quello di colonna.

Una matrice è identificata dal suo nome, che può essere un singolo carattere alfabetico, o una stringa di caratteri; gli indici vengono scritti in piccolo in basso a de-

stra del nome della matrice. Nei linguaggi di programmazione come COBOL, FORTRAN o BASIC, dove questa rappresentazione non è possibile, gli indici sono scritti tra parentesi; in Pascal e in APL si usano le parentesi quadre. Perciò, l'espressione X[2,3] in un programma Pascal indica l'elemento che si trova nella seconda riga e nella terza colonna di una matrice di nome X.

Una matrice che ha lo stesso numero di righe e di colonne si dice quadrata. La diagonale principale di una matrice quadrata contiene gli elementi x_{11}, x_{22}, x_{33}, ..., x_{nn} e talvolta viene indicata come semplicemente la diagonale. Una matrice quadrata che contiene tutti 1 nella diagonale principale, e 0 in tutte le altre posizioni si dice matrice unitaria. Per esempio:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

è una matrice unitaria 4×4.

Ora che abbiamo introdotto la terminologia di base per le matrici, possiamo proseguire con lo studio delle principali operazioni aritmetiche su matrici, e con la loro realizzazione in Pascal.

Aritmetica delle matrici

Cominciamo col definire la matrice trasposta, poi studieremo il prodotto scalare, l'addizione, la sottrazione e la moltiplicazione.

La matrice trasposta

Si ottiene scambiando le righe con le colonne di una matrice. L'elemento x_{ij} diventa l'elemento x_{ji} della matrice trasposta, che viene indicata con la lettera X^T. Così, se

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

allora

$$X^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Notate che la trasposta di una matrice quadrata si ottiene ruotando la matrice originale intorno alla diagonale principale.

Due matrici sono uguali se ogni elemento dell'una è uguale al corrispondente elemento dell'altra, cioè X = Y se, per ogni elemento

$$x_{ij} = y_{ij}$$

Una matrice quadrata si dice simmetrica se è uguale alla sua trasposta; in questo caso ogni elemento x_{ij} è uguale all'elemento x_{ji}.

Prodotto di una matrice per uno scalare

Se si moltiplica una matrice X per uno scalare s, ogni elemento della matrice risulta moltiplicato per s. La seguente istruzione Pascal crea una matrice Y come risultato del prodotto tra una matrice X e uno scalare S.

```
FOR I : = 1 TO N DO
  FOR J: = 1 TO M DO
    Y[I,J] : = X[I,J]*S
```

Addizione e sottrazione tra matrici

Una matrice può essere sommata o sottratta ad un'altra matrice se entrambe hanno lo stesso numero di righe e di colonne. La somma delle matrici X e Y produce una matrice Z, che si rappresenta come:

$$Z = X + Y$$

Così, se:

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix}$$

e

$$Y = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \end{bmatrix}$$

allora

$$Z = \begin{bmatrix} x_{11} + y_{11} & x_{12} + y_{12} & x_{13} + y_{13} \\ x_{21} + y_{21} & x_{22} + y_{22} & x_{23} + y_{23} \end{bmatrix}$$

Le istruzioni Pascal corrispondenti sono:

```
FOR I : = 1 TO N DO
  FOR J : = 1 TO M DO
    Z[I,J] : = X[I,J] + Y[I,J]
```

Ogni elemento di Z è formato dalla somma dei corrispondenti elementi di X e Y.
Analogamente, la sottrazione di una matrice da un'altra

$$Z = X - Y$$

viene eseguita sottraendo ogni elemento della seconda matrice dal corrispondente elemento della prima.

Moltiplicazione tra matrici

Una matrice può essere moltiplicata per un'altra se il numero delle righe della prima è uguale al numero delle colonne della seconda; in questo caso le matrici si dicono conformi. Dunque, se X è una matrice che contiene m righe e n colonne, e Y un'altra matrice che contiene n righe e p colonne, il prodotto:

$$Z = X\ Y$$

produce una matrice Z con m righe e p colonne, cioè con lo stesso numero di righe della matrice X e lo stesso numero di colonne della matrice Y.

Ogni elemento di Z è formato da una somma di prodotti: gli elementi di una riga della matrice X sono moltiplicati ognuno per il corrispondente elemento della colonna di Y, e i prodotti vengono sommati.

Data una matrice X

$$X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix}$$

e

$$Y = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \end{bmatrix}$$

la matrice Z = X Y sarà

$$Z = \begin{bmatrix} x_{11}y_{11} + x_{12}y_{21} & x_{11}y_{12} + x_{12}y_{22} & x_{11}y_{13} + x_{12}y_{23} \\ x_{21}y_{11} + x_{22}y_{21} & x_{21}y_{12} + x_{22}y_{22} & x_{21}y_{13} + x_{22}y_{23} \\ x_{31}y_{11} + x_{32}y_{21} & x_{31}y_{12} + x_{32}y_{22} & x_{31}y_{13} + x_{32}y_{23} \end{bmatrix}$$

Ogni elemento jk di Z è formato dalla riga j della matrice X e dalla colonna k della matrice Y, secondo lo schema:

$$z_{jk} = x_{j1}y_{1k} + x_{j2}y_{2k} + \ldots + x_{jn}y_{nk}$$

così, se

$$X = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

e

$$Y = \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

il prodotto

$$Z = XY = \begin{bmatrix} 47 & 52 & 57 \\ 64 & 71 & 78 \\ 81 & 90 & 99 \end{bmatrix}$$

è una matrice quadrata 3X3; il primo elemento, z_{11}, è calcolato come:

$$(1)(7) + (4)(10) = 47$$

Il prodotto tra matrici non gode della proprietà commutativa, cioè il prodotto YX non è in genere uguale al prodotto XY. Invertendo l'ordine dei fattori, nell'esempio precedente, si ottiene una matrice 2x2 invece che 3x3:

$$YX = \begin{bmatrix} 50 & 122 \\ 68 & 167 \end{bmatrix}$$

Notate che il prodotto scalare tra due vettori segue le regole della moltiplicazione tra matrici se il primo è un vettore colonna (cioè ha una sola colonna) e il secondo è un vettore riga (cioè ha una sola riga).

Abbiamo visto come il Pascal tratta vettori e matrici mediante array mono e bidimensionali; siamo ora in grado di scrivere un programma con le istruzioni che abbiamo imparato.

PROGRAMMA PASCAL: MOLTIPLICAZIONE TRA MATRICI

Nel Capitolo 4 avremo bisogno di una routine che esegua la moltiplicazione tra matrici, per moltiplicare la trasposta della matrice X per l'originale, e il vettore y per la matrice x. Scriveremo ora questa routine.

Il programma della figura 3.1 contiene la procedura SQUARE che esegue entrambe le operazioni; il programma principale genera la matrice X e il vettore y, e la procedura SQUARE calcola la matrice A e il vettore g in base alle equazioni:

$$X^T X = A \text{ e } yX = g$$

```
PROGRAM MATR1( OUTPUT);
(* 8 Feb 81 *)
(* Programma Pascal che esegue *)
(* la moltiplicazione tra matrici *)

CONST
   RMAX = 9;
   CMAX = 3;

TYPE
   ARY   = ARRAY[1..RMAX] OF REAL;
   ARYS  = ARRAY[1.:CMAX] OF REAL;
   ARY2  = ARRAY[1..RMAX, 1..CMAX] OF REAL;
   ARY2S = ARRAY[1..CMAX, 1..CMAX] OF REAL;
```

Figura 3.1 — Programma di moltiplicazione tra matrici ($A = X^T X$, $G = Y X$).

```
VAR
   Y : ARY;
   G : ARYS;
   X : ARY2;
   A : ARY2S;
   NROW, NCOL : INTEGER;

PROCEDURE GET_DATA(VAR X : ARY2;
                   VAR Y : ARY;
            VAR NROW, NCOL : INTEGER);

(* riceve i valori per nrow, ncol e le matrici x, y *)

VAR  I, J : INTEGER;

BEGIN
   NROW := 5;
   NCOL := 3;

   FOR I := 1 TO NROW DO
      BEGIN
         X[I,1] := 1;
         FOR J := 2 TO NCOL DO
            X[I,J] := I * X[I,J-1];
         Y[I] := 2 * I
      END
END (* procedure get_data *);

PROCEDURE WRITE_DATA;

(* stampa le risposte *)
```

Figura 3.1 — Programma di moltiplicazione tra matrici ($A = X^T X$, $G = Y X$) (segue).

```
VAR
   I, J : INTEGER;


BEGIN
   WRITELN;
   WRITELN('            X                    Y');
   FOR I := 1 TO NROW DO
      BEGIN
         FOR J := 1 TO NCOL DO
            WRITE( X[I,J]:7:1, ' ');
         WRITELN( ' : ', Y[I] :7:1)
      END;
   WRITELN('            A                    G');
   FOR I := 1 TO NCOL DO
      BEGIN
         FOR J := 1 TO NCOL DO
            WRITE( A[I,J]:7:1, ' ');
         WRITELN( ' : ', G[I] :7:1)
      END
END (* write_data *);


PROCEDURE SQUARE(X : ARY2;
                 Y  : ARY;
            VAR A : ARY2S;
            VAR G : ARYS;
      NROW,NCOL  : INTEGER);

(* routine di moltiplicazione di matrici *)
(* A = Transpone X volte X *)
(* G = Y volte X *)
```

Figura 3.1 — Programma di moltiplicazione tra matrici ($A = X^T X$, $G = Y X$) (segue).

```
VAR
   I, K, L : INTEGER;

BEGIN (* square *)
   FOR K := 1 TO NCOL DO
      BEGIN
         FOR L := 1 TO K DO
            BEGIN
               A[K,L] := 0;
               FOR I := 1 TO NROW DO
                  BEGIN
                     A[K,L] := A[K,L] + X[I,L] * X[I,K];
                     IF K <> L THEN A[L,K] := A[K,L]
                  END
            END (* ciclo L *)
         G[K] := 0;
         FOR I := 1 TO NROW DO
            G[K] := G[K] + Y[I] * X[I,K]
      END (* ciclo K *)
END (* square *);

BEGIN (* programma principale *)
   GET_DATA (X, Y, NROW, NCOL);
   SQUARE(X, Y, A, G, NROW, NCOL);
   WRITE_DATA
END.
```

Figura 3.1 — Programma di moltiplicazione tra matrici ($A = X^T X$, $G = Y X$) (segue).

La matrice X contiene 5 righe e 3 colonne, e il vettore y contiene 5 elementi.

Poichè la matrice risultante A è simmetrica, il calcolo può essere semplificato, in quanto ci saranno elementi come:

A[1,3] : = A[3,1]

La matrice A avrà 3 righe (come la trasposta di X) e 3 colonne (come X); il vettore g avrà 3 elementi. In effetti, sia y che g devono essere considerati come vettori riga, cioè con una riga e 5 colonne; altrimenti, se vogliamo considerarli come vettori colonna, l'equazione di moltiplicazione dovrà essere scritta come:

$$\mathbf{y}^T X = \mathbf{g}^T$$

dove il trasposto del vettore colonna diventa vettore riga.

Scrivete il programma della figura 3.1 ed eseguitelo; i risultati si presenteranno come in figura 3.2.

```
          X                          Y
     1.0       1.0       1.0          2.0
     1.0       2.0       4.0          4.0
     1.0       3.0       9.0          6.0
     1.0       4.0      16.0          8.0
     1.0       5.0      25.0         10.0
          A                          G
     5.0      15.0      55.0         30.0
    15.0      55.0     225.0        110.0
    55.0     225.0     979.0        450.0
```

Figura 3.2 — Risultati del programma di moltiplicazione tra matrici.

DETERMINANTI

Il determinante di una matrice quadrata X si indica con |X|, ed è un valore scalare Per una matrice 2x2, l'elemento in alto a sinistra viene moltiplicato per quello in basso a destra, e dal risultato viene sottratto il prodotto dell'elemento in basso a sinistra per quello in alto a destra:

$$|X| = x_{11}x_{22} - x_{12}x_{21}$$

Per esempio, il determinante della matrice:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

è −2.

Il determinante di matrici con dimensioni maggiori si ottiene moltiplicando ogni e-

lemento della prima riga per il determinante della matrice che si ottiene eliminando la riga e la colonna dell'elemento in questione. Una definizione ricorsiva è:

$$|X| = x_{11}s_{11} - x_{12}s_{12} + x_{13}s_{13} - \ldots (-1)^{n+1}x_{1n}s_{1n}$$

dove x_{11}, x_{12}, sono gli elementi della prima riga della matrice X e s_{1n} è il determinante della matrice che si ottiene eliminando la riga 1 e la colonna n. Il determinante della matrice:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

è uguale a:

$$1 \begin{vmatrix} 5 & 6 \\ 8 & 0 \end{vmatrix} - 2 \begin{vmatrix} 4 & 6 \\ 7 & 0 \end{vmatrix} + 3 \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix}$$

Il passo successivo consiste nel calcolare i minori:

$$\begin{vmatrix} 5 & 6 \\ 8 & 0 \end{vmatrix} \quad \begin{vmatrix} 4 & 6 \\ 7 & 0 \end{vmatrix} \quad \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix}$$

secondo la procedura:

$$1(5 \cdot 0 - 8 \cdot 6) - 2(4 \cdot 0 - 7 \cdot 6) + 3(4 \cdot 8 - 7 \cdot 5)$$

Il valore del determinante è, in questo caso, 27. Se ogni elemento di una riga o di una colonna è 0, il determinante è 0. Se due righe o due colonne sono identiche, il determinante è 0.

Abbiamo visto come calcolare il determinante di una matrice. Poichè useremo i determinanti nei prossimi capitoli, studiamo ora un programma per calcolarli.

PROGRAMMA PASCAL: DETERMINANTI

In figura 3.3 c'è un programma per calcolare il determinante di una matrice 3x3. La procedura GET_DATA chiede all'utente di inserire gli elementi della matrice, e la

procedura DETER calcola il determinante. Scrivete il programma ed eseguitelo. Gli elementi della matrice sono inseriti riga per riga, cioè nell'ordine:

A[1,1], A[1,2], A[1,3], A[2,1], A[2,2], ...

alla fine compare sul video il valore del determinante, seguito dalla domanda «Ancora?»; rispondete Y se volete calcolare il determinante di un'altra matrice.

```
PROGRAM DETERM(INPUT, OUTPUT);
(* 8 Feb 81 *)

(* Programma Pascal che calcola *)
(* il determinante di una matrice 3-per-3 *)

TYPE
  ARY2 = ARRAY[1..3, 1..3] OF REAL;

VAR
  A      : ARY2;
  N      : INTEGER;
  YESNO : CHAR;
  D      : REAL;

PROCEDURE GET_DATA(VAR A : ARY2;
                   VAR N : INTEGER);
(* riceve i valori di n e delle matrici x, y *)

VAR
  I, J : INTEGER;

BEGIN
  N := 3;
  WRITELN;
```

Figura 3.3 — Determinante di una matrice 3 × 3.

```
  FOR I := 1 TO N DO
    BEGIN
      FOR J := 1 TO N DO
        BEGIN
          WRITE( J:3, ': ');
          READLN( A[I,J])
        END (* ciclo j *)
    END (* ciclo i *)
  WRITELN;
  FOR I:= 1 TO N DO
    BEGIN
      FOR J:= 1 TO N DO
        WRITE( A[I,J] :7:4, ' ');
      WRITELN
    END;
  WRITELN
END (* procedure get_data *);

FUNCTION DETER (A : ARY2) : REAL;

(* calcola il determinante di una matrice 3-per-3 *)

VAR
  SUM : REAL;

BEGIN
  SUM := A[1,1] * (A[2,2]*A[3,3] - A[3,2]*A[2,3])
         - A[1,2] * (A[2,1]*A[3,3] - A[3,1]*A[2,3])
         + A[1,3] * (A[2,1]*A[3,2] - A[3,1]*A[2,2]);
  DETER := SUM
END;
```

Figura 3.3 — Determinante di una matrice 3 × 3 (segue).

```
BEGIN (* programma principale *)
  REPEAT
    GET_DATA (A, N);
    D := DETER(A);
    WRITELN('Il determinante è', D);
    WRITELN;
    WRITE('Ancora?');
    READLN( YESNO)
  UNTIL (YESNO <> 'Y' ) AND (YESNO <> 'y')
END.
```

Figura 3.3 — Determinante di una matrice 3 × 3 (segue).

MATRICE INVERSA E DIVISIONE TRA MATRICI

L'inversa di una matrice non singolare si scrive come

$$X^{-1}$$

L'inversa di una matrice singolare non è definita. Il prodotto di una matrice per la sua inversa è la matrice unitaria:

$$XX^{-1} = I$$

L'inversione di matrice è analoga alle altre operazioni di inversione, poichè l'inversa di una matrice inversa è la matrice originale.

$$(X^{-1})^{-1} = X$$

Quando l'operazione viene effettuata con un programma su di un calcolatore, può capitare che la precedente formula non sia verificata a causa degli errori di arrotondamento.

La divisione tra matrici viene eseguita con una combinazione di inversioni e molti-

plicazioni. Se dobbiamo dividere la matrice X per la matrice Y, eseguiamo dapprima un'inversione su Y, e moltiplichiamo il risultato per X. L'operazione viene scritta come:

$$XY^{-1}$$

L'inversione di matrici può essere usata per trovare la soluzione di un sistema di equazioni lineari. Se abbiamo una matrice di coefficienti A e un vettore costante y, vogliamo trovare un vettore b tale che:

$$A\mathbf{b} = \mathbf{y}$$

La soluzione si ottiene dal prodotto dell'inversa della matrice A per il vettore costante y (in questo ordine).

$$A^{-1}\mathbf{y} = \mathbf{b}$$

Poichè la soluzione dei sistemi di equazioni lineari è l'argomento del Capitolo 4, rimandiamo a quel punto lo studio di un programma per implementarla.

SOMMARIO

Abbiamo visto come rappresentare in Pascal vettore e matrici, e come eseguire operazioni aritmetiche su di essi. Abbiamo studiato due importanti programmi, uno per eseguire la moltiplicazione tra matrici, e l'altro per calcolare i determinanti. Useremo questi programmi nei prossimi capitoli.

CAPITOLO 4

RISOLUZIONE DI SISTEMI DI EQUAZIONI LINEARI

INTRODUZIONE

In questo Capitolo studieremo la risoluzione di un sistema di equazioni lineari, con la regola di Cramer, il metodo di eliminazione di Gauss, il metodo di eliminazione di Gauss-Jordan e quello di Gauss-Seidel. Inoltre affronteremo il problema dei sistemi mal-condizionati generando un insieme di matrici di Hilbert. Svilupperemo anche dei programmi Pascal su questi argomenti, tra cui uno per risolvere un sistema di equazioni con diversi vettori di termini noti e con coefficienti complessi; infine presenteremo un programma che produce la "miglior curva interpolante" di un sistema predeterminato.

Cominceremo col descrivere le equazioni lineari e un semplice sistema di equazioni.

EQUAZIONI LINEARI E RISOLUZIONE DI SISTEMI

Un'equazione lineare è una somma di termini del tipo:

$$Ax + By + Cz = D \quad .$$

In questa equazione x, y e z sono variabili mentre A, B, C e D sono costanti. In ogni termine non può comparire più di una variabile, alla prima potenza. Espressioni come:

$$2x^2 + 3y^2 = 4$$
$$\sin(x) + \log(y) = 9$$

e

$$xy = 2$$

sono equazioni non lineari nelle variabili x e y.

Un'equazione del tipo

$$\frac{x}{A} + y \log (B) = p$$

è lineare nelle variabili x e y, mentre diventa non lineare se consideriamo A e B come variabili e x, y e p come termini noti.

Equazioni lineari si incontrano molto spesso in tutti i rami della scienza e dell'ingegneria, ed è quindi necessario conoscere dei metodi efficaci di risoluzione. Quando ci sono diverse incognite e uno stesso numero di equazioni indipendenti, esiste un'unica soluzione. In questo capitolo studieremo diversi metodi di risoluzione di un sistema di equazioni lineari; i metodi di soluzione di equazioni non lineari sono discussi in altri capitoli.

Le due equazioni lineari:

$$\begin{aligned} x - 2y &= 1 \\ 2x + y &= 7 \end{aligned}$$

rappresentano due rette nel piano x-y; la soluzione di questo sistema rappresenta l'intersezione tra le due rette, e una soluzione grafica del problema potrebbe ottenersi tracciando il grafico delle rette e trovando le coordinate del punto di intersezione.

La prima equazione ha un'intercetta sull'asse y di −0.5 e una tangente di 0.5, come si vede riscrivendo l'equazione nella forma:

$$y = -0.5 + 0.5x$$

La seconda equazione ha un'intercetta sull'asse y di 7 e una tangente di −2:

$$y = 7 - 2x$$

Le due rette si intersecano nel punto:

$$x = 3, \; y = 1$$

che rappresenta la soluzione del sistema; possiamo eseguire una verifica sostituendo i valori di x e y precedenti nelle equazioni originali.

Un altro metodo è quello di moltiplicare la seconda equazione per 2 e sommarla alla prima. L'equazione risultante contiene solo la variabile x, e può quindi essere risolta direttamente.

$$\begin{array}{rcl} x - 2y &=& 1 \\ 4x + 2y &=& 14 \\ \hline 5x \qquad &=& 15 \quad \text{o} \quad x = 3 \end{array}$$

Il valore di x può ora essere sostituito in una delle equazioni originali per trovare il corrispondente valore di y. Entrambi questi metodi risolvono il sistema delle due equazioni, ma sono in genere noiosi da applicare per un sistema con più equazioni.

Nel prossimo paragrafo studieremo un metodo di risoluzione più sofisticato, che fa uso di matrici e vettori.

LA REGOLA DI CRAMER

È un metodo comodo per risolvere sistemi di due o tre equazioni; è particolarmente potente per risolvere il sistema a mano o con un calcolatore tascabile. Le equazioni devono essere scritte con tutte le incognite da una parte dell'uguale, e i termini noti dall'altra. I termini contenenti la stessa incognita sono allineati in verticale. Le due equazioni del paragrafo precedente erano inizialmente scritte così.

Con i coefficienti delle incognite si forma una matrice che prende il nome di matrice dei coefficienti; i termini noti formano invece un vettore a parte. Con le nostre due equazioni avremmo una matrice A e un vettore z.

$$A = \begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix} \qquad \mathbf{z} = \begin{bmatrix} 1 \\ 7 \end{bmatrix}$$

La soluzione è data dalle seguenti equazioni:

$$x = \frac{D_1}{D} \qquad y = \frac{D_2}{D}$$

dove D è il determinante della matrice dei coefficienti:

$$D = \begin{vmatrix} 1 & -2 \\ 2 & 1 \end{vmatrix} = (1)(1) - (2)(-2) = 5$$

e D_1 è il determinante della matrice che si ottiene sostituendo gli elementi del vettore z nella prima colonna della matrice, corrispondente all'incognita x:

$$D_1 = \begin{vmatrix} 1 & -2 \\ 7 & 1 \end{vmatrix} = (1)(1) - (7)(-2) = 15$$

Analogamente D_2 si ottiene sostituendo il vettore z nella seconda colonna della matrice:

$$D_2 = \begin{vmatrix} 1 & 1 \\ 2 & 7 \end{vmatrix} = (1)(7) - (2)(1) = 5$$

La soluzione è:

$$x = \frac{15}{5} = 3 \qquad y = \frac{5}{5} = 1$$

Naturalmente alcuni sistemi di equazioni hanno più di una soluzione; vedremo ora come risolverli con la regola di Cramer.

Dipendenza lineare

Se il determinante della matrice dei coefficienti è zero, non esiste un'unica soluzione. Questa situazione si verifica quando le equazioni sono linearmente dipendenti, cioè una di esse può essere ottenuta da una combinazione delle altre. In questo caso la matrice si dice singolare. Come esempio di dipendenza lineare, consideriamo le due equazioni:

$$\begin{aligned} x + y &= 5 \\ 2x + 2y &= 2 \end{aligned}$$

La matrice dei coefficienti è:

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$$

con determinante zero. In effetti le due equazioni rappresentano due rette parallele, che, ovviamente, non hanno punti di intersezione. Non esiste dunque soluzione.

Consideriamo un altro esempio:

$$\begin{aligned} x + y &= 5 \\ 2x + 2y &= 10 \end{aligned}$$

la matrice dei coefficienti di queste due equazioni è uguale alla precedente, ma in questo caso le due rette si sovrappongono, e hanno quindi un numero infinito di punti di intersezione; il sistema ammette infinite soluzioni.

Consideriamo ora delle applicazioni di questo metodo alla risoluzione di sistemi di due o tre equazioni lineari. Nel prossimo paragrafo risolveremo un problema con un programma in Pascal che abbiamo presentato nel Capitolo 3.

Esempio: Un circuito elettrico in corrente continua

Un interessante problema pratico, che comporta la soluzione di un sistema di equazioni, si presenta nello studio di un circuito elettrico. Consideriamo la rete di resistenze e di generatori di tensione continua della figura 4.1, che ha quattro nodi e sei rami. Sul lato sinistro della rete c'è un generatore di tensione di 20 volt, su quello destro un generatore di 5 volt; inoltre ci sono sei resistenze di valore noto.

Si vogliono determinare i valori delle correnti che circolano nei vari rami, e quelli delle cadute di tensione ai capi delle resistenze. I primi si trovano risolvendo un sistema di sei equazioni lineari, oppure, semplificando il problema considerando le correnti che circolano nelle tre maglie, e risolvendo quindi un sistema di tre equazioni; poi la corrente di ogni ramo si ricava da quella della maglia di cui il ramo fa parte.

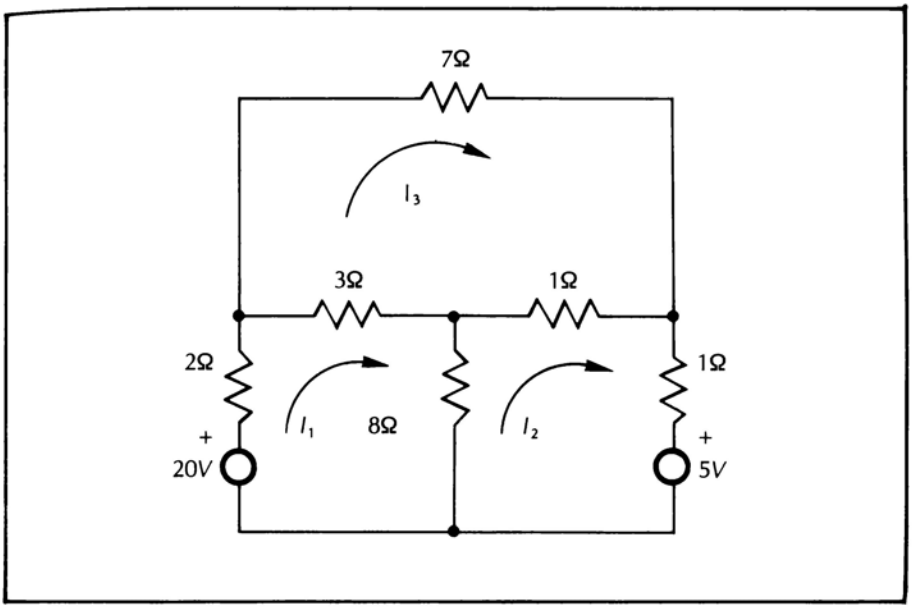


Figura 4.1 — Un circuito con resistenze e generatori di tensione continua.

La corrente nella maglia in basso a sinistra è indicata con I_1, quella in basso a destra con I_2, e quella in alto con I_3. Le equazioni delle tre maglie si ricavano dalla legge di Kirchoff, considerando positive le correnti che percorrono la maglia in senso orario.

$$13\,I_1 - 8\,I_2 - 3\,I_3 - 20 = 0 \quad \text{(maglia in basso a sinistra)}$$
$$-8\,I_1 + 10\,I_2 - I_3 + 5 = 0 \quad \text{(maglia in basso a destra)}$$
$$-3\,I_1 - I_2 + 11\,I_3 = 0 \quad \text{(maglia in alto)}$$

La matrice dei coefficienti e il vettore dei termini noti corrispondenti sono:

$$\begin{bmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \end{bmatrix} \qquad \begin{bmatrix} 20 \\ -5 \\ 0 \end{bmatrix}$$

Applicando la regola di Cramer troviamo:

$$D = \begin{vmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \end{vmatrix} \qquad D_1 = \begin{vmatrix} 20 & -8 & -3 \\ -5 & 10 & -1 \\ 0 & -1 & 11 \end{vmatrix}$$

$$D_2 = \begin{vmatrix} 13 & 20 & -3 \\ -8 & -5 & -1 \\ -3 & 0 & 11 \end{vmatrix} \qquad D_3 = \begin{vmatrix} 13 & -8 & 20 \\ -8 & 10 & -5 \\ -3 & -1 & 0 \end{vmatrix}$$

Questi determinanti si calcolano usando il programma DETERM della figura 3.3 del Capitolo precedente; le correnti risultanti sono:

$$I_1 = \frac{D_1}{D} = \frac{1725}{575} = 3 \text{ amps}$$

$$I_2 = \frac{D_2}{D} = \frac{1150}{575} = 2 \text{ amps}$$

$$I_3 = \frac{D_3}{D} = \frac{575}{575} = 1 \text{ amp}$$

Il programma DETERM ci dà dei risultati corretti, ma non è semplice da usare; per ognuno dei quattro determinanti bisogna inserire nove numeri, per un totale di 36 dati.

Studieremo adesso un programma realizzato per applicare la regola di Cramer alla risoluzione di sistemi di equazioni lineari.

PROGRAMMA PASCAL: UN ELEGANTE USO DELLA REGOLA DI CRAMER

Il programma della figura 4.2 semplifica notevolmente la procedura di risoluzione; le operazioni matematiche in realtà sono le stesse di prima, ma in questa versione si devono inserire solo dodici numeri, nove per i coefficienti e tre per il vettore dei termini noti.

Esecuzione del programma

Scrivete il programma della figura 4.2 ed eseguitelo; i coefficienti e il termine noto della prima equazione sono inseriti sulla prima linea; i corrispondenti valori della seconda equazione sono sulla seconda linea, e quelli della terza sulla terza linea. Lasciate uno spazio dopo ogni coefficiente, e premete RETURN dopo il termine noto. I dati di ingresso si presenteranno così:

```
Soluzione di un sistema di equazioni con la regola di Cramer

Equazione 1
1 :  13     2 :  -8     3 :  -3   ,  C :  20

Equazione 2
1 :  -8     2 :  10     3 :  -1   ,  C :  -5

Equazione 3
1 :  -3     2 :  -1     3 : -11   ,  C :  0
```

I dati di ingresso verranno poi nuovamente stampati, seguiti dalla soluzione:

```
 13.0000     -8.0000     -3.0000     : 20.000
 -8.0000     10.0000     -1.0000     : -5.0000
 -3.0000     -1.0000     11.0000     :  0.0000

  3.00000     2.00000     1.00000
```

A questo punto apparirà la domanda

Ancora?

a cui dovete rispondere Y se volete risolvere un altro sistema, N in caso contrario.

I dati di ingresso possono anche essere inseriti uno per riga, premendo RETURN dopo ogni dato; in questo caso si presenteranno nel modo seguente:

```
Equazione 1
1 :  13
2 :  -8
3 :  -3
C :  20

Equazione 2
1 :  -8
2 :  10
3 :  -1
C :  -5

Equazione 3
1 :  -3
2 :  -1
3 :  11
C :  0
```

IL METODO DI ELIMINAZIONE DI GAUSS

La regola di Cramer è un buon metodo di risoluzione dei sistemi con due o tre equazioni lineari, soprattutto quando devono essere risolti "a mano". D'altra parte, il tempo di calcolo aumenta con la quarta potenza della dimensione della matrice; risolvere un sistema di 6 equazioni richiederà, dunque, un tempo 16 volte maggiore di quello necessario per un sistema di 3 equazioni. Il metodo di eliminazione di Gauss è più efficace; il tempo di calcolo cresce con la terza potenza del numero di equazioni, e quindi per risolvere un sistema di sei equazioni ci vorrà un tempo 8 volte superiore a quello necessario per un sistema di tre equazioni.

Il metodo di eliminazione di Gauss si può facilmente implementare su di un calcolatore, mentre non si presta alla risoluzione manuale, poichè usa una tecnica leggermente complicata, che richiede molte moltiplicazioni, divisioni e sottrazioni; questo implica un'inevitabile perdita di precisione che impone un limite al numero di equazioni del sistema. Studiamo ora passo passo questo metodo.

Il metodo di Gauss

Le equazioni originali sono manipolate in modo che la matrice dei coefficienti contenga un 1 in ogni posizione della diagonale principale, e uno 0 in tutte le posizioni in basso a sinistra.

```
PROGRAM SIMQ1(INPUT,OUTPUT);
(* 12 Feb 81 *)
(* Programma Pascal che risolve un *)
(* sistema di tre equazioni con la regola di Cramer *)

CONST
   RMAX = 3;
   CMAX = 3;

TYPE
   ARYS  = ARRAY[1..CMAX] OF REAL;
   ARY2S = ARRAY[1..RMAX, 1..CMAX] OF REAL;

VAR
   Y, COEF : ARYS;
   A       : ARY2S;
   N       : INTEGER;
   YESNO   : CHAR;
   ERROR   : BOOLEAN;

PROCEDURE GET_DATA(VAR A : ARY2S;
                   VAR Y : ARYS;
                   VAR N : INTEGER);

(* riceve i valori per n e per le matrici a, y *)

VAR
   I, J : INTEGER;

BEGIN (* procedure get_data *)
   WRITELN;
   N := RMAX;
```

Figura 4.2 — Risoluzione di un sistema di tre equazioni lineari con la regola di Cramer.

```
  FOR I := 1 TO N DO
    BEGIN
      WRITELN ('Equazione', 1:3);
      FOR J := 1 TO N DO
        BEGIN
          WRITE( J:3, ': ');
          READ( A[I,J])
        END;
      WRITE( ', C: ');
      READLN (Y[I])
    END;
  WRITELN;
  FOR I:= 1 TO N DO
    BEGIN
      FOR J:= 1 TO N DO
        WRITE( A[I,J] :7:4, ' ');
      WRITELN( ' : ', Y[I] :7:4)
    END;
  WRITELN
END (* procedure get_data *);

PROCEDURE WRITE_DATA;
(* stampa i risultati *)

VAR
  I : INTEGER;

BEGIN (* write_data *)
  FOR I := 1 TO N DO
    WRITE( COEF[I] :9:5);
  WRITELN
END (* write_data *);
```

Figura 4.2 — Risoluzione di un sistema di tre equazioni lineari con la regola di Cramer (segue).

```
PROCEDURE SOLVE( A : ARY2S;
                 Y : ARYS;
      VAR     COEF : ARYS;
                 N : INTEGER;
      VAR   ERROR : BOOLEAN);

VAR
  B    : ARY2S;
  I, J : INTEGER;
  DET  : REAL;

FUNCTION DETER (A : ARY2S) : REAL;

(* calcola il determinante di una matrice 3-per-3 *)

VAR
  SUM : REAL;

BEGIN (* function deter *)
  SUM := A[1,1] * (A[2,2]*A[3,3] - A[3,2]*A[2,3])
       - A[1,2] * (A[2,1]*A[3,3] - A[3,1]*A[2,3])
       + A[1,3] * (A[2,1]*A[3,2] - A[3,1]*A[2,2]);
  DETER := SUM
END (* function deter *);

PROCEDURE SETUP (VAR   B : ARY2S;
                 VAR  COEF : ARYS;
                        J : INTEGER);

VAR
  I : INTEGER;
```

Figura 4.2 — Risoluzione di un sistema di tre equazioni lineari con la regola di Cramer (segue).

```
BEGIN (* setup *)
   FOR I := 1 TO N DO
      BEGIN
         B[I,J] := Y[I];
         IF J > 1 THEN B[I,J-1] := A[I,J-1]
      END;
   COEF[J] := DETER(B)/DET
END (* setup *);

BEGIN (* procedure solve *)
   ERROR := FALSE;
   FOR I := 1 TO N DO
      FOR J := 1 TO N DO
         B[I,J] := A[I,J];
   DET := DETER(B);
   IF DET = 0.0 THEN
      BEGIN
         ERROR := TRUE;
         WRITELN('ERRORE: matrice singolare')
      END
   ELSE
      BEGIN
         SETUP( B, COEF, 1);
         SETUP( B, COEF, 2);
         SETUP( B, COEF, 3)
      END (* else *)
END (* procedure solve *);
```

Figura 4.2 — Risoluzione di un sistema di tre equazioni lineari con la regola di Cramer (segue).

```
BEGIN (* programma principale *)
  WRITELN;
  WRITELN('Soluzione di un sistema con la regola di Cramer');
  REPEAT
    GET_DATA (A, Y, N);
    SOLVE (A, Y, COEF, N, ERROR);
    IF NOT ERROR THEN WRITE_DATA;
    WRITELN;
    WRITE('Ancora?');
    READLN( YESNO)
  UNTIL (YESNO <> 'Y' ) AND (YESNO <> 'y')
END.
```

Figura 4.2 — Risoluzione di un sistema di tre equazioni lineari con la regola di Cramer (segue).

Questo metodo usa fondamentalmente due tipi di operazioni su matrici: la moltiplicazione per uno scalare e la addizione. Ogni equazione può essere moltiplicata per una costante, senza cambiare il risultato; ciò equivale a moltiplicare tutti gli elementi di una riga della matrice dei coefficienti, e l'elemento corrispondente del vettore dei termini noti, per lo stesso valore. Ogni equazione può anche essere sostituita dalla somma di due equazioni.

Nelle pagine seguenti mostreremo l'uso del metodo di eliminazione di Gauss sulle equazioni che abbiamo ricavato dal circuito elettrico di figura 4.1. Inizialmente abbiamo una matrice dei coefficienti e un vettore dei termini noti come quelli seguenti:

$$\begin{bmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \end{bmatrix} \cdot \begin{bmatrix} 20 \\ -5 \\ 0 \end{bmatrix}$$

La prima variabile viene eliminata da tutte le equazioni tranne la prima, e le equazioni vengono manipolate in modo da avere 1 come primo elemento, e 0 in tutti gli altri, nella prima colonna. L'operazione viene eseguita nel modo seguente: tutti gli elementi della prima riga vengono divisi per il primo (il pivot), ottenendo così un 1 nella prima posizione della diagonale principale. La prima equazione diventa:

$$\begin{bmatrix} 1 & -0.61 & -0.23 \end{bmatrix} \qquad \begin{bmatrix} 1.5 \end{bmatrix}$$

Con una combinazione delle prime due righe, la prima incognità è eliminata dalla seconda colonna, precisamente moltiplicando la prima riga per il primo elemento della seconda, e sottraendola poi dalla seconda; la nuova seconda riga è ora:

$$\begin{bmatrix} 0 & 5.1 & -2.8 \end{bmatrix} \qquad \begin{bmatrix} 7.3 \end{bmatrix}$$

Analogamente, la prima variabile viene eliminata dalla terza riga; a questo punto le tre equazioni sono:

$$\begin{bmatrix} 1 & -0.61 & -0.23 \\ 0 & 5.1 & -2.8 \\ 0 & -2.8 & 10.3 \end{bmatrix} \qquad \begin{bmatrix} 1.5 \\ 7.3 \\ 4.6 \end{bmatrix}$$

Il passo successivo consiste nel produrre un 1 nella seconda posizione della seconda riga; gli elementi della seconda riga sono divisi per il secondo elemento. Poi la seconda incognita viene eliminata dalla terza equazione generando uno 0 nella seconda posizione, proprio sotto il pivot. Le tre equazioni ora si presentano come:

$$\begin{bmatrix} 1 & -0.61 & -0.23 \\ 0 & 1 & -0.56 \\ 0 & 0 & 8.7 \end{bmatrix} \qquad \begin{bmatrix} 1.5 \\ 1.4 \\ 8.7 \end{bmatrix}$$

L'ultimo passo di questa fase consiste nell'ottenere un 1 nella terza posizione della terza equazione, dividendo tutti gli elementi per il terzo:

$$\begin{bmatrix} 1 & -0.61 & -0.23 \\ 0 & 1 & -0.56 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1.5 \\ 1.4 \\ 1 \end{bmatrix}$$

Le tre equazioni originali si sono trasformate in:

$$\begin{aligned} x - 0.61y - 0.23z &= 1.5 \\ y - 0.56z &= 1.4 \\ z &= 1 \end{aligned}$$

La terza equazione può essere risolta immediatamente:

$$z = 1$$

La seconda equazione diventa:

$$y = 1.4 + 0.56z$$

Sostituendo il valore di z, si trova $y = 2$; sostituendo y e z nella prima equazione si trova $x = 3$. Questo metodo di calcolo prende il nome di "sostituzione a ritroso".

Una maggior precisione col metodo di Gauss

La precisione del metodo di eliminazione di Gauss può essere migliorata scambiando fra di loro le righe, in modo che l'elemento col maggiore valor assoluto diventi il pivot. Supponiamo per esempio che le tre equazioni precedenti siano scritte in un altro ordine:

$$\begin{bmatrix} -3 & -1 & 11 \\ 13 & -8 & -3 \\ -8 & 10 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 \\ 20 \\ -5 \end{bmatrix}$$

La prima equazione dovrebbe essere divisa per –3 per ottenere 1 nella prima posizione; ma il risultato sarà più preciso se scambiamo tra loro la prima e la seconda equazione, in modo da avere 13 come pivot. Dopo avere eliminato la prima incognita dalla seconda e dalla terza equazione, si ottiene:

$$\begin{bmatrix} 1 & -0.61 & -0.23 \\ 0 & -2.8 & 10.3 \\ 0 & 5.1 & -2.8 \end{bmatrix} \qquad \begin{bmatrix} 1.5 \\ 4.6 \\ 7.3 \end{bmatrix}$$

A questo punto si potrebbero scambiare tra loro la seconda e la terza equazione, per avere un numero maggiore (5.1) come pivot.

C'è un altro motivo per cui conviene scambiare tra loro le righe: se compare uno zero sulla diagonale principale, la riga non può essere divisa per il pivot. Scambiando la riga con una seguente lo zero non occuperà più la posizione del pivot.

Ora che abbiamo seguito passo passo il metodo di eliminazione di Gauss, per la verità un po' noioso da applicare a mano, sapremo apprezzare l'eleganza del programma Pascal del prossimo paragrafo.

PROGRAMMA PASCAL: IL METODO DI ELIMINAZIONE DI GAUSS

Il programma di figura 4.3 risolve un sistema di equazioni lineari (otto al massimo) con il metodo di eliminazione di Gauss; il numero di equazioni può essere aumentato modificando le variabili MAXR e MAXC, definite all'inizio del programma. Scrivetelo ed eseguitelo.

Esecuzione del programma

Questo programma è simile al precedente, ma comincia col chiedere il numero di equazioni; rispondete con un numero da 2 a 8, e premete RETURN. Inserite i coefficienti di ogni equazione e i termini noti, un'equazione per riga; lasciate uno spazio dopo ogni numero e premete RETURN alla fine di ogni linea. Inserite i valori del sistema di equazioni precedente e controllate che i risultati siano [3 2 1].

L'uscita del programma si presenterà come nella figura 4.4.

```
PROGRAM GAUS(INPUT, OUTPUT);
(12 Feb 81)

(* Programma Pascal che esegue la soluzione *)
(* di un sistema per mezzo della eliminazione Gaussiana *)
(* la procedura GAUSS è inclusa *)

CONST
  MAXR = 8;
  MAXC = 8;

TYPE
  ARY   = ARRAY[1..MAXR] OF REAL;
  ARYS  = ARRAY[1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXR, 1..MAXC] OF REAL;

VAR
  Y     : ARYS;
  COEF  : ARYS;
  A     : ARY2S;
  N, M  : INTEGER;
  ERROR : BOOLEAN;

PROCEDURE GET_DATA(VAR A : ARY2S;
                   VAR Y : ARYS;
                   VAR N, M : INTEGER);

(* riceve i valori per n e per le matrici a e y *)

VAR
  I, J : INTEGER;
```

Figura 4.3 — Risoluzione di sistemi con il metodo di eliminazione di Gauss.

```
BEGIN
  WRITELN;
  REPEAT
    WRITE ('Quante equazioni?');
    READLN( N);
    M := N
  UNTIL N < MAXR;
  IF N > 1 THEN
    BEGIN
      FOR I := 1 TO N DO
        BEGIN
          WRITELN ('Equazione', 1:3);
          FOR J := 1 TO N DO
            BEGIN
              WRITE( J:3, ': ');
              READ( A[I,J])
            END;
          WRITE( ', C: ');
          READ (Y[I]);
          READLN (* linea bianca *)
        END;
      WRITELN;
      FOR I:= 1 TO N DO
        BEGIN
          FOR J:= 1 TO M DO
            WRITE( A[I,J] :7:4, ' ');
          WRITELN( ' : ', Y[I] :7:4)
        END;
      WRITELN
    END (* if n>1 *)
END (* procedure get_data *);
```

Figura 4.3 — Risoluzione di sistemi con il metodo di eliminazione di Gauss (segue).

```
PROCEDURE WRITE_DATA;
(* stampa i risultati *)

VAR
  I : INTEGER;

BEGIN
  FOR I := 1 TO M DO
    WRITE( COEF[I] :9:5);
  WRITELN
END (* write_data *);

PROCEDURE GAUSS
                (A    : ARY2S;
                 Y    : ARYS;
          VAR   COEF : ARYS;
                NCOL : INTEGER;
          VAR  ERROR : BOOLEAN);

(* soluzione di una matrice per mezzo della Eliminazione Gaussiana *)
(* 8 Feb 81 *)
(* Adattata da Gilder *)

VAR
  B    : ARY2S (* matrice di lavoro, nrow,ncol *);
  W    : ARYS  (* matrice di lavoro, ncol long *);
  I, J, I1, K, L,
  N    : INTEGER;
  HOLD, SUM, T, AB, BIG : REAL;

BEGIN
  ERROR := FALSE;
  N := NCOL;
```

Figura 4.3 — Risoluzione di sistemi con il metodo di eliminazione di Gauss (segue).

```
FOR I := 1 TO N DO
  BEGIN (* copia nelle matrici di lavoro *)
    FOR J := 1 TO N DO
      B[I,J] := A[I,J];
    W[I] := Y[I]
  END;
FOR I := 1 TO N - 1 DO
  BEGIN
    BIG := ABS(B[I,I]);
    L := I;
    I1 := I + 1;
    FOR J := I1 TO N DO
      BEGIN (* ricerca l'elemento maggiore *)
        AB := ABS(B[J,I]);
        IF AB > BIG THEN
          BEGIN
            BIG := AB;
            L := J
          END
      END;
    IF BIG = 0.0 THEN ERROR := TRUE
    ELSE
      BEGIN
        IF L <> I THEN
          BEGIN
            (* scambia le righe per porre *)
            (* l'elemento maggiore sulla diagonale *)
            FOR J := 1 TO N DO
              BEGIN
                HOLD := B[L,J];
                B[L,J] := B[I,J];
                B[I,J] := HOLD
              END;
```

Figura 4.3 — Risoluzione di sistemi con il metodo di eliminazione di Gauss (segue).

```
                HOLD := W[L];
                W[L] := W[I];
                W[I] := HOLD
              END (* if L <> i *);
            FOR J := I1 TO N DO
              BEGIN
                T := B[J,I]/B[I,I];
                FOR K := I1 TO N DO
                  B[J,K] := B[J,K] - T * B[I,K];
                W[J] := W[J] - T * W[I]
              END (* ciclo j *)
        END (* if big *)
      END; (* ciclo i *)
    IF B[N,N] = 0.0 THEN ERROR := TRUE
    ELSE
      BEGIN
        COEF[N] := W[N]/B[N,N];
        I := N - 1;
        (* sostituzione inversa *)
        REPEAT
          SUM := 0.0;
          FOR J := I+1 TO N DO
            SUM := SUM + B[I,J] * COEF[J];
          COEF[I] := (W[I] - SUM)/B[I,I];
          I := I - 1
        UNTIL I = 0
      END (* IF b[n,n] = 0 *);
    IF ERROR THEN WRITELN('ERRORE: Matrice singolare')
  END (* gauss *);

  BEGIN (* programma principale *)
    WRITELN;
    WRITELN('Soluzione di un sistema con l'eliminazione di Gauss');
```

Figura 4.3 — **Risoluzione di sistemi con il metodo di eliminazione di Gauss (segue).**

```
    REPEAT
      GET_DATA (A, Y, N, M);
      IF N > 1 THEN
        BEGIN
          GAUSS (A, Y, COEF, N, ERROR);
          IF NOT ERROR THEN WRITE_DATA
        END
    UNTIL N < 2
  END.
```

Figura 4.3 — Risoluzione di sistemi con il metodo di eliminazione di Gauss (segue).

```
Soluzione di un sistema con l'eliminazione di Gauss

Quante equazioni? 3
Equazione  1
 1: 13    2: -8    3: -3 , C: 20
Equazione  2
 1: -8    2: 10    3: -1 , C: -5
Equazione  3
 1: -3    2: -1    3: 11 , C: 0

13.0000   -8.0000   -3.0000    :  20.0000
-8.0000   10.0000   -1.0000    : -5.0000
-3.0000   -1.0000   11.0000    :  0.0000

 3.00000   2.00000   1.00000
```

Figura 4.4 — Risultati del programma di risoluzione del circuito elettrico con il metodo di eliminazione di Gauss.

Dopo avere trovato le soluzioni, il programma riparte dall'inizio; questa volta troverete la soluzione delle seguenti tre equazioni, e ne prenderete nota perchè ci servirà in seguito.

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & -1 \\ 3 & 1 & -3 \end{bmatrix} \qquad \begin{bmatrix} 6 \\ 1 \\ -4 \end{bmatrix}$$

Riprenderemo questo sistema alla fine del prossimo paragrafo.
La terza volta inserite due righe uguali, per esempio:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 2 & 1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 6 \\ 6 \\ 1 \end{bmatrix}$$

Il programma dovrebbe stampare un messaggio di errore per segnalarvi che la matrice è singolare. Per uscire dal programma inserite uno zero o un valore negativo come numero di equazioni.

IL METODO DI ELIMINAZIONE DI GAUSS-JORDAN

È una variante del metodo di eliminazione di Gauss; presenta più o meno gli stessi vantaggi e svantaggi di questo. Il tempo di esecuzione cresce con la terza potenza della dimensione della matrice dei coefficienti, e ci sono molte moltiplicazioni, divisioni e sottrazioni che ne riducono la precisione; inoltre usa un algoritmo più complicato di quello di Gauss; ciononostante questo metodo è in genere il più conveniente, ed è quello che useremo nei prossimi capitoli. Infatti, consente di determinare facilmente il vettore soluzione con l'inversa della matrice dei coefficienti. Studiamo ora questo metodo passa passo.

Spiegazione dettagliata del metodo di Gauss-Jordan

Gli elementi della diagonale principale sono trasformati in 1 come nel metodo di Gauss, e tutti gli elementi sopra e sotto la diagonale principale sono trasformati in 0; la matrice dei coefficienti diventa una matrice unitaria, e il vettore dei termini noti diventa la soluzione. Nel caso del circuito elettrico della figura 4.1, l'insieme finale dei valori diventa:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

che corrisponde alle tre equazioni:

$$\begin{aligned} x \qquad\qquad &= 3 \\ y \qquad &= 2 \\ z &= 1 \end{aligned}$$

che hanno come soluzione

$$x = 3,\ y = 2,\ z = 1$$

Supponiamo di mettere una matrice unitaria a destra della matrice del sistema di equazioni originale; se eseguiamo sulla prima tutte le stesse operazioni che eseguiamo sugli elementi dell'altra, alla fine essa sarà diventata l'inversa della matrice originale dei coefficienti. L'insieme di valori:

$$\begin{bmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \end{bmatrix} \qquad \begin{bmatrix} 20 \\ -5 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

si trasformerà in:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} 0.19 & 0.16 & 0.07 \\ 0.16 & 0.23 & 0.06 \\ 0.07 & 0.06 & 0.11 \end{bmatrix}$$

La matrice sulla destra è l'inversa dell'originale. In realtà non è necessario usare due matrici diverse per queste operazioni; la matrice inversa può occupare fisicamente lo stesso spazio di quella originale e, alla fine del calcolo, sostituirsi ad essa, mentre il vettore con la soluzione prenderà il posto di quello originale con i termini noti. Così, se la matrice dei coefficienti e il vettore dei termini noti originali sono:

$$\begin{bmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \end{bmatrix} \qquad \begin{bmatrix} 20 \\ -5 \\ 0 \end{bmatrix}$$

la matrice inversa e il vettore soluzione occuperanno alla fine gli stessi array:

$$\begin{bmatrix} 0.19 & 0.16 & 0.07 \\ 0.16 & 0.23 & 0.06 \\ 0.07 & 0.06 & 0.11 \end{bmatrix} \qquad \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

Presenteremo ora un programma Pascal che realizza questo algoritmo abbastanza complesso.

PROGRAMMA PASCAL: IL METODO DI ELIMINAZIONE DI GAUSS-JORDAN

Il programma della figura 4.5 risolve sistemi di equazioni lineari con il metodo di Gauss-Jordan. Il programma principale chiama la procedura esterna GAUSSJ. Se il vostro compilatore non consente procedure esterne, dovrete inserirla nel programma principale.

La procedura GAUSSJ è mostrata nella figura 4.6; in questa versione viene restituita la matrice inversa al posto di quella originale, B; per questo la matrice originale A è duplicata nel programma principale prima di chiamare GAUSSJ. Invece, il vettore dei termini noti, Y, è separato da quello della soluzione, COEF.

Scrivete il programma ed eseguitelo. Inserite le equazioni del circuito elettrico e verificate che la soluzione è [3 2 1]. Poi inserite i dati seguenti:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & -1 \\ 3 & 1 & -3 \end{bmatrix} \qquad \begin{bmatrix} 6 \\ 1 \\ -4 \end{bmatrix}$$

Notate che la risposta, in questo caso, non è la stessa di quella trovata con il metodo di Gauss, ma l'insieme dei valori:

$$x = 1,\ y = 2,\ z = 3$$

è anch'esso una soluzione del sistema. Come può essere? Una delle equazioni è una combinazione lineare delle altre due; precisamente, la terza è uguale alla seconda moltiplicata per due meno la prima, e la matrice dei coefficienti è singolare. Se forniamo queste tre equazioni, come dati, al programma che applica la regola di Cramer del capitolo precedente, questo fatto viene segnalato immediatamente.

Purtroppo questo tipo di dipendenza lineare è difficile da scoprire; infatti il determinante della matrice dei coefficienti può essere molto piccolo, ma diverso da zero a causa degli errori di arrotondamento che si accumulano con il metodo di eliminazione. Per fortuna, però, se risolviamo il sistema con diversi algoritmi, avremo risultati diversi; perciò, se ci sono problemi di dipendenza lineare, è meglio risolvere il sistema con almeno due metodi diversi.

Abbiamo visto come questo programma tratta una matrice dei coefficienti con il relativo vettore dei termini noti; ora studieremo come modificarlo per trattare in modo opportuno una stessa matrice dei coefficienti con diversi vettori dei termini noti. Riprendiamo il caso del circuito elettrico, e insieme continuiamo la discussione sulle matrici dei coefficienti inverse e su come usarle per risolvere i sistemi.

```
PROGRAM SOLVGJ(INPUT, OUTPUT);
(* 12 Feb 81 *)

(* Programma Pascal che esegue la soluzione di un sistema *)
(* per mezzo della eliminazione di Gauss-Jordan *)

CONST
  MAXR = 8;
  MAXC = 8;

TYPE
  ARY   = ARRAY[1..MAXR] OF REAL;
  ARYS  = ARRAY[1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXR, 1..MAXC] OF REAL;

VAR
  Y        : ARYS;
  COEF     : ARYS;
  A, B     : ARY2S;
  N, M, I, J: INTEGER;
  ERROR    : BOOLEAN;

PROCEDURE GET_DATA(VAR A : ARY2S;
                   VAR Y : ARYS;
                   VAR N, M : INTEGER);

(* riceve i valori per n e per le matrici a, y *)

VAR
  I, J : INTEGER;
```

Figura 4.5 — Risoluzione di sistemi di equazioni con il metodo di eliminazione di Gauss-Jordan.

```
BEGIN
  WRITELN;
  REPEAT
    WRITE('Quante equazioni?');
    READLN( N);
    M := N
  UNTIL N < MAXR;
  IF N > 1 THEN
    BEGIN
      FOR I := 1 TO N DO
        BEGIN
          WRITELN('Equazione', 1:3);
          FOR J := 1 TO N DO
            BEGIN
              WRITE( J:3, ': ');
              READ( A[I,J])
            END;
          WRITE(', C: ');
          READLN (Y[I]) (* linea bianca *)
        END;
      WRITELN;
      FOR I:= 1 TO N DO
        BEGIN
          FOR J:= 1 TO M DO
            WRITE( A[I,J] :7:4, ' ');
          WRITELN( ' : ', Y[I] :7:4)
        END;
      WRITELN
    END (* if n > 1 *)
END (* procedure get_data *);

PROCEDURE WRITE_DATA;
(* stampa i risultati *)
```

Figura 4.5 — Risoluzione di sistemi di equazioni con il metodo di eliminazione di Gauss-Jordan (segue).

```
VAR
   I : INTEGER;

BEGIN
   FOR I := 1 TO M DO
      WRITE( COEF[I] :9:5);
   WRITELN
END (* write_data *);

(* PROCEDURE gaussj
   (VAR b          : ary2s;
    y              : arys;
    VAR coef       : arys;
    ncol           : integer;
    VAR error      : boolean);
extern; *)
(*$F GAUSSJ.PAS *)

BEGIN (* programma principale *)
   WRITELN;
   WRITELN
('Soluzione di un sistema con l'eliminazione di Gauss-Jordan');
   REPEAT
      GET_DATA (A, Y, N, M);
      IF N > 1 THEN
         BEGIN
            FOR I := 1 TO N DO
               FOR J := 1 TO N DO
                  B[I,J] := A[I,J] (* carica la matrice di lavoro *);
            GAUSSJ (B, Y, COEF, N, ERROR);
            IF NOT ERROR THEN WRITE_DATA
         END
   UNTIL N < 2
END.
```

Figura 4.5 — Risoluzione di sistemi di equazioni con il metodo di eliminazione di Gauss-Jordan (segue).

```
PROCEDURE GAUSSJ
  (VAR B       : ARY2S; (* matrice quadrata dei coefficienti *)
   Y           : ARYS; (* vettore delle costanti *)
   VAR COEF    : ARYS; (* vettore delle soluzioni *)
   NCOL        : INTEGER; (* ordine della matrice *)
   VAR ERROR : BOOLEAN); (* vero se la matrice è singolare *)

(*  Inversione e soluzione di matrice secondo Gauss Jordan *)
(*  Adattata da McCormick *)
(*  8 Feb 81 *)
(*    B(N,N) matrice dei coefficienti, diventa inversa *)
(*    Y(N) vettore delle costanti originali *)
(*    W(N,M) vettore costanti. diventa vettore delle soluzioni *)
(*    DETERM è il determinante *)
(*    ERROR = 1 se singolare *)
(*    INDEX(N,3) *)
(*    NV è il numero dei vettori delle costanti *)

LABEL 99;

VAR
  W     : ARRAY[1..MAXC, 1..MAXC] OF REAL;
  INDEX : ARRAY[1..MAXC, 1..3] OF INTEGER;
  I, J, K, L, NV, IROW, ICOL, N, L1        : INTEGER;
  DETERM, PIVOT, HOLD, SUM, T, AB, BIG : REAL;

PROCEDURE SWAP(VAR A, B: REAL);

VAR
  HOLD: REAL;
```

Figura 4.6 — Procedura di Gauss-Jordan

```
BEGIN (* swap *)
   HOLD := A;
   A := B;
   B := HOLD
END (* procedure swap *);

PROCEDURE GAUSJ2;

LABEL 98;

VAR
   I, J, K, L, L1: INTEGER;

PROCEDURE GAUSJ3;

VAR
   L: INTEGER;

BEGIN (* procedure gausj3 *)
   (* scambia le righe per porre il pivot sulla diagonale *)
   IF IROW <> ICOL THEN
      BEGIN
         DETERM := - DETERM;
         FOR L := 1 TO N DO
            SWAP(B[IROW, L], B[ICOL, L]);
         IF NV > 0 THEN
            FOR L := 1 TO NV DO
               SWAP(W[JROW, L], W[ICOL, L])
      END (* if irow <> icol *)
END (* gausj3 *);
```

Figura 4.6 — Procedura di Gauss-Jordan (segue).

```
BEGIN (* procedure gausj2 *)
  (* inizio effettivo di gaussj *)
  ERROR := FALSE;
  NV := 1 (* vettore delle costanti singole *);
  N := NCOL;
  FOR I := 1 TO N DO
    BEGIN
      W[I, 1] := Y[I] (* copia il vettore delle costanti *);
      INDEX[I, 3] := 0
    END;
  DETERM := 1.0;
  FOR I := 1 TO N DO
    BEGIN
      (* ricerca l'elemento maggiore *)
      BIG := 0.0;
      FOR J := 1 TO N DO
        BEGIN
          IF INDEX[J, 3] <> 1 THEN
            BEGIN
              FOR K := 1 TO N DO
                BEGIN
                  IF INDEX[K, 3] > 1 THEN
                    BEGIN
                      WRITELN ('ERRORE: matrice singolare');
                      ERROR := TRUE;
                      GOTO 98        (* aborto *)
                    END;
                  IF INDEX[K, 3] < 1 THEN
                    IF ABS(B[J, K]) > BIG THEN
```

Figura 4.6 — Procedura di Gauss-Jordan (segue).

```
                    BEGIN
                       IROW := J;
                       ICOL := K;
                       BIG := ABS(B[J, K])
                    END
              END (* ciclo k *)
        END
   END (* ciclo j *)

INDEX[ICOL, 3] := INDEX[ICOL, 3] + 1;
INDEX[I, 1] := IROW;
INDEX[I, 2] := ICOL;
GAUSJ3 (* ulteriore suddivisione di gaussj *);
(* divide la riga pivot per la colonna pivot *)
PIVOT := B[ICOL, ICOL];
DETERM := DETERM * PIVOT;
B[ICOL, ICOL] := 1.0;

FOR L := 1 TO N DO
   B[ICOL, L] := B[ICOL, L]/PIVOT;
IF NV > 0 THEN
   FOR L := 1 TO NV DO
      W[ICOL, L] := W[ICOL, L]/PIVOT;
(* converte le righe non pivot *)

FOR L1 := 1 TO N DO
   BEGIN
      IF L1 <> ICOL THEN
         BEGIN
            T := B[L1, ICOL];
            B[L1, ICOL] := 0.0;
            FOR L := 1 TO N DO
               B[L1, L] := B[L1, L] - B[ICOL, L] * T;
            IF NV > 0 THEN
```

Figura 4.6 — Procedura di Gauss-Jordan (segue).

```
                    FOR L := 1 TO NV DO
                        W[L1, L] := W[L1, L] - W[ICOL, L] * T;
                END (* IF l1 <> icol *)
            END
        END (* ciclo i *);
    98:
END (* gausj2 *);

BEGIN     (* programma principale Gauss-Jordan *)
    GAUSJ2 (* prima metà di gaussj *);
    IF ERROR THEN GOTO 99;
    (* scambia le colonne *)
    FOR I := 1 TO N DO
        BEGIN
            L := N - I + 1;
            IF INDEX[L, 1] <> INDEX[L, 2] THEN
                BEGIN
                    IROW := INDEX[L, 1];
                    ICOL := INDEX[L, 2];
                    FOR K := 1 TO N DO
                        SWAP(B[K, IROW], B[K, ICOL])
                END (* if index *)
        END (* ciclo i *);
    FOR K := 1 TO N DO
        IF INDEX[K, 3] <> 1 THEN
            BEGIN
                WRITELN ('ERRORE: matrice singolare');
                ERROR := TRUE;
                GOTO 99  (* aborto *)
            END;
    FOR I := 1 TO N DO
        COEF[I] := W[I, 1];
    99:
END (* procedura gaussj *);
```

Figura 4.6 — Procedura di Gauss-Jordan (segue).

VETTORI DI TERMINI NOTI MULTIPLI E INVERSIONE DELLE MATRICI

Nel capitolo precedente abbiamo visto che la soluzione di un sistema di equazioni lineari si può trovare moltiplicando l'inverso della matrice dei coefficienti per il vettore dei termini noti. Se chiamiamo A la matrice, e y il vettore, il vettore soluzione b sarà:

$$\mathbf{b} = A^{-1} \ \mathbf{y}$$

In questo capitolo, invece, abbiamo sviluppato dei metodi (regola di Cramer, metodo di eliminazione di Gauss e di Gauss-Jordan) che determinano la soluzione direttamente, senza utilizzare l'inversa della matrice dei coefficienti.

Può capitarci di dover risolvere diversi sistemi di equazioni con la stessa matrice dei coefficienti, e diversi vettori dei termini noti. In questo caso, possiamo invertire la matrice dei coefficienti, e ottenere le diverse soluzioni dal prodotto della matrice inversa per i vettori dei termini noti. Anche in questo caso, però, è in genere più veloce il metodo di eliminazione di Gauss-Jordan, applicato simultaneamente alla matrice e a tutti i vettori. In effetti, la procedura che implementa il metodo di Gauss-Jordan nel paragrafo precedente è studiata per questi casi.

Consideriamo per esempio il circuito della figura 4.1, e supponiamo di voler determinare le correnti delle tre maglie per tre diverse configurazioni:

1. il circuito originale
2. il circuito con il generatore di 5 volt invertito
3. il circuito con entrambi i generatori di tensione invertiti

Queste tre diverse configurazioni corrispondono alle tre seguenti equazioni:

$$\begin{bmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \end{bmatrix} \qquad \begin{bmatrix} 20 \\ -5 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \end{bmatrix} \quad . \quad \begin{bmatrix} 20 \\ 5 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \end{bmatrix} \qquad \begin{bmatrix} -20 \\ 5 \\ 0 \end{bmatrix}$$

Tutti e tre i sistemi possono essere risolti separatamente con una delle tecniche viste prima, ma è meglio risolverli contemporaneamente con la procedura di Gauss-Jordan. La matrice dei coefficienti è unica per tutti e tre i circuiti, mentre il vettore

dei termini noti diventa una matrice, in cui ogni colonna rappresenta una configurazione diversa e ne produrrà la soluzione. Le matrici da passare alla routine di Gauss-Jordan sono:

$$\begin{bmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \end{bmatrix} \qquad \begin{bmatrix} 20 & 20 & -20 \\ -5 & 5 & 5 \\ 0 & 0 & 0 \end{bmatrix}$$

e la routine restituirà al programma principale i seguenti valori:

$$\begin{bmatrix} 0.19 & 0.16 & 0.07 \\ 0.16 & 0.23 & 0.06 \\ 0.07 & 0.06 & 0.11 \end{bmatrix} \qquad \begin{bmatrix} 3 & 4.58 & -3 \\ 2 & 4.33 & -2 \\ 1 & 1.64 & -1 \end{bmatrix}$$

La matrice di sinistra, che inizialmente conteneva i coefficienti, ora contiene l'inversa; quella di destra, che inizialmente conteneva i vettori dei termini noti, ora contiene le soluzioni. Le risposte per i tre diversi circuiti sono:

	Soluzione		
Circuito	I_1	I_2	I_3
1	3	2	1
2	4.58	4.33	1.64
3	−3	−2	−1

Nella versione precedente avevamo copiato la matrice dei coefficienti originale A in un array di lavoro B prima di chiamare la procedura di Gauss-Jordan; inoltre, all'inizio di questa, il vettore dei termini noti Y veniva copiato nella prima colonna della matrice W. Alla fine della procedura, la soluzione veniva copiata dalla prima colonna della matrice W nel vettore COEF. Ora studieremo una versione modificata del programma.

PROGRAMMA PASCAL: SECONDA VERSIONE DEL METODO DI GAUSS-JORDAN

Il programma della figura 4.8 è un'altra versione del metodo di eliminazione di Gauss-Jordan. Sono stati cambiati i parametri formali della procedura di Gauss-Jordan, e l'inversa della matrice dei coefficienti viene restituita nell'array che inizialmente conteneva la matrice originale dei coefficienti, come prima; in questa versione, però, la matrice soluzione viene restituita nell'array che conteneva i termini noti, e inoltre viene restituito il determinante della matrice dei coefficienti.

Esecuzione del programma

Scrivete il programma ed eseguitelo. Come prima, vi chiederà il numero delle equazioni, e, successivamente, il numero dei vettori dei termini noti. Se rispondete 1

il programma si comporterà esattamente come prima, con l'unica differenza che il vettore soluzione sarà stampato verticalmente invece che orizzontalmente.

Se il numero dei vettori è maggiore di 1, i termini noti di ogni equazione devono essere inseriti sulla stessa linea dell'equazione. Per esempio, se vogliamo risolvere contemporaneamente i tre circuiti precedenti con questo programma la prima riga di dati sarà:

13 –8 –3 20 20 –20

Risolveteli e verificate che i risultati siano esatti.

Questa nuova versione prevede anche il caso che il numero dei vettori dei termini noti sia zero e che quindi sia inserita solo una matrice dei coefficienti. Il programma in questo caso stamperà l'inversa e il determinante della matrice dei coefficienti.

Inserite la matrice seguente, che già conoscete:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & -1 \\ 3 & 1 & -3 \end{bmatrix}$$

Questa matrice è singolare e quindi ha determinante zero, ma il programma di Gauss-Jordan potrebbe dare un valore piccolo, ma diverso da zero. Un esempio di uscita del programma potrebbe essere il seguente, dove viene inserito 0 per uscire:

```
Soluzione di un sistema con Gauss-Jordan
Vettori delle costanti multiple, o matrice inversa

Quante equazioni? 3
Quanti vettori di costanti? 0
Equazione  1
 1: 1    2: 1    3: 1
Equazione  2
 1: 2    2: 1    3: -1
Equazione  3
 1: 3    2: 1    3: -3

              Matrice
1.0000    1.0000    1.0000
2.0000    1.0000   -1.0000
3.0000    1.0000   -3.0000

       Inversa
-11184800.0    22369600.0   -11184800.0
 16777200.0   -33554400.0    16777200.0
 -5592400.0    11184800.0    -5592410.0

 Il determinante è   1.78814e-7

 Quante equazioni? 0
```

Figura 4.7 — Risultati del secondo programma del metodo di Gauss-Jordan.

```
PROGRAM SOLVGV(INPUT, OUTPUT);
(* 8 Feb 81 *)
(* Programma Pascal che esegue la soluzione di un sistema *)
(* per mezzo della eliminazione di Gauss-Jordan *)
(* con i vettori delle costanti multiple *)

CONST
  MAXR = 7;
  MAXC = 7;

TYPE
  ARY2S = ARRAY[1..MAXR, 1..MAXC] OF REAL;

VAR
  A, Y      : ARY2S;
  N, NVEC   : INTEGER;
  ERROR     : BOOLEAN;
  DETERM    : REAL;

PROCEDURE GET_DATA(VAR A : ARY2S;
                   VAR Y : ARY2S;
                   VAR N, NVEC : INTEGER);
(* riceve i valori per n, nvec e le matrici a, y *)

VAR
  I, J : INTEGER;

BEGIN
  WRITELN;
  REPEAT
    WRITE ('Quante equazioni?');
    READLN( N)
  UNTIL N < MAXR;
```

Figura 4.8 — Risoluzione di sistemi e inversione di matrici con il metodo di Gauss-Jordan

```
IF N > 1 THEN
  BEGIN
    WRITE (´Quanti vettori delle costanti?´);
    READLN( NVEC);
    FOR I := 1 TO N DO
      BEGIN
        FOR J := 1 TO N DO
          BEGIN
            WRITE(', ', J:3, ': ');
            READ( A[I,J])
          END;
        IF NVEC > 0 THEN
          BEGIN
            FOR J := 1 TO NVEC DO
              BEGIN
                WRITE( ', C: ');
                READ(Y[I,J])
              END;
            READLN
          END
      END (* ciclo I *);
    WRITELN;
    WRITE( '          Matrice´);
    IF NVEC > 0 THEN WRITE('          Costanti´);
    WRITELN;
    FOR I:= 1 TO N DO
      BEGIN
        FOR J:= 1 TO N DO
          WRITE( A[I,J] :7:4, ' ');
        FOR J := 1 TO NVEC DO
          WRITE( ' :', Y[I,J] :7:4);
        WRITELN
      END (* ciclo i *);
```

Figura 4.8 — Risoluzione di sistemi e inversione di matrici con il metodo di Gauss-Jordan (segue).

```
        WRITELN
      END (* if n > 1 *)
END (* procedure get_data *);

PROCEDURE WRITE_DATA;
(* stampa i risultati *)

VAR
  I, J : INTEGER;
BEGIN
  IF NVEC > 0 THEN
    BEGIN
      WRITELN ('Soluzione');
      FOR I := 1 TO N DO
        BEGIN
          FOR J := 1 TO NVEC DO
            WRITE( Y[I,J] :9:5);
          WRITELN
        END
    END (* IF *)
  ELSE
    BEGIN
      WRITELN('    (Inverso');
      FOR I := 1 TO N DO
        BEGIN
          FOR J := 1 TO N DO
            WRITE( A[I,J] :9:5);
          WRITELN
        END;
      WRITELN;
      WRITE ('Il Determinante è', DETERM)
    END (* ELSE *);
  WRITELN
END (* write_data *);
```

Figura 4.8 — Risoluzione di sistemi e inversione di matrici con il metodo di Gauss-Jordan (segue).

```
PROCEDURE GAUSJV
   (VAR B        : ARY2S; (* matrice quadrate dei coefficienti *)
    VAR W        : ARY2S; (* matrice dei vettori costanti *)
    VAR DETERM : REAL;  (* il determinante *)
        NCOL     : INTEGER;  (* ordine della matrice *)
        NV       : INTEGER;  (* numero di costanti *)
    VAR ERROR  : BOOLEAN); (* vero se la matrice è singolare *)

(*  soluzione e inversione della matrice Gauss-Jordan *)
(*  21 Nov 81 *)
(*   B(N,N) matrice dei coefficienti, diviene inversa *)
(*   W(N,M) vettore/i delle costanti diventano vettori soluzione *)
(*   DETERM è il determinante *)
(*   ERROR = 1 se singolare *)
(*   INDEX(N,3) *)
(*   NV è il numero dei vettori delle costanti *)

LABEL 99;

VAR
   INDEX: ARRAY[1..MAXC, 1..3] OF INTEGER;
   I, J, K, L, IROW, ICOL, N, L1     : INTEGER;
   PIVOT, HOLD, SUM, T, AB, BIG      : REAL;

PROCEDURE SWAP(VAR A, B: REAL);
VAR
   HOLD: REAL;
BEGIN (* swap *)
   HOLD := A;
   A := B;
   B := HOLD
END (* procedure swap *);
```

Figura 4.8 — Risoluzione di sistemi e inversione di matrici con il metodo di Gauss-Jordan (segue).

```
PROCEDURE GAUSJ2;

LABEL 98;

VAR
  I, J, K, L, L1: INTEGER;

PROCEDURE GAUSJ3;

VAR
  L: INTEGER;

BEGIN (* procedure gausj3 *)
  (* scambia le righe per porre la pivot sulla diagonale *)
  IF IROW <> ICOL THEN
    BEGIN
      DETERM := - DETERM;
      FOR L := 1 TO N DO
        SWAP(B[IROW, L], B[ICOL, L]);
      IF NV > 0 THEN
        FOR L := 1 TO NV DO
      SWAP(W[IROW, L], W[ICOL, L])
    END     (* if irow <> icol *)
END (* gausj3 *);

BEGIN (* inizio effettivo di gaussj *)
  (* actual start of gaussj *)
  ERROR := FALSE;
  N := NCOL;
  FOR I := 1 TO N DO
    INDEX[I, 3] := 0;
  DETERM := 1.0;
```

Figura 4.8 — Risoluzione di sistemi e inversione di matrici con il metodo di Gauss-Jordan (segue).

```
FOR I := 1 TO N DO
  BEGIN
    (* ricerca l'elemento maggiore *)
    BIG := 0.0;
    FOR J := 1 TO N DO
      BEGIN
        IF INDEX[J, 3] <> 1 THEN
          BEGIN
            FOR K := 1 TO N DO
              BEGIN
                IF INDEX[K, 3] > 1 THEN
                  BEGIN
                    WRITELN ('ERRORE: matrice singolare');
                    ERROR := TRUE;
                    GOTO 98 (* aborto *)
                  END;
                IF INDEX[K, 3] < 1 THEN
                  IF ABS(B[J, K]) > BIG THEN
                    BEGIN
                      IROW := J;
                      ICOL := K;
                      BIG := ABS(B[J, K])
                    END
              END (* ciclo k *)
          END (* IF *)
      END (* ciclo j *)
    INDEX[ICOL, 3] := INDEX[ICOL, 3] + 1;
    INDEX[I, 1] := IROW;
    INDEX[I, 2] := ICOL;
    GAUSJ3 (* ulteriore suddivisione di gaussj *);
    (* divide la riga pivot per la colonna pivot *)
    PIVOT := B[ICOL, ICOL];
    DETERM := DETERM * PIVOT;
```

Figura 4.8 — Risoluzione di sistemi e inversione di matrici con il metodo di Gauss-Jordan (segue).

```
        B[ICOL, ICOL] := 1.0;
        FOR L := 1 TO N DO
          B[ICOL, L] := B[ICOL, L]/PIVOT;
        IF NV > 0 THEN
          FOR L := 1 TO NV DO
            W[ICOL, L] := W[ICOL, L]/PIVOT;
        (* converte le righe non pivot *)
        FOR L1 := 1 TO N DO
          BEGIN
            IF L1 <> ICOL THEN
              BEGIN
                T := B[L1, ICOL];
                B[L1, ICOL] := 0;
                FOR L := 1 TO N DO
                  B[L1, L] := B[L1, L] - B[ICOL, L] * T;
                IF NV > 0 THEN
                  FOR L := 1 TO NV DO
                    W[L1, L] := W[L1, L] - W[ICOL, L] * T
              END  (* IF L1 <> icol *)
          END (* FOR L1 *)
      END (* ciclo i *);
  98:
END (* gausj2 *);

BEGIN   (* Programma principale Gauss-Jordan *)
  (* prima metà di gaussj *);
  IF ERROR THEN GOTO 99;
  (* scambio delle colonne *)
  FOR I := 1 TO N DO
    BEGIN
      L := N - I + 1;
```

Figura 4.8 — Risoluzione di sistemi e inversione di matrici con il metodo di Gauss-Jordan (segue).

```
        IF INDEX[L, 1] <> INDEX[L, 2] THEN
          BEGIN
            IROW := INDEX[L, 1];
            ICOL := INDEX[L, 2];
            FOR K := 1 TO N DO
              SWAP(B[K, IROW], B[K, ICOL])
          END (* if index *)
      END (* ciclo i *);
    FOR K := 1 TO N DO
      IF INDEX[K, 3] <> 1 THEN
        BEGIN
          WRITELN ('ERRORE: matrice singolare');
          ERROR := TRUE;
          GOTO 99 (* aborto *)
        END;
    99:
  END (* procedura gaussj *);

  BEGIN (* programma principale *)
    WRITELN;
    WRITELN ('Soluzione simultanea per mezzo di Gauss-Jordan');
    WRITELN ('Vettori delle costanti multiple, o matrice inversa');
    REPEAT
      GET_DATA (A, Y, N, NVEC);
      IF N > 1 THEN
        BEGIN
          GAUSJV (A, Y, DETERM, N, NVEC, ERROR);
          IF NOT ERROR THEN WRITE_DATA
        END
    UNTIL N < 2
  END.
```

Figura 4.8 — Risoluzione di sistemi e inversione di matrici con il metodo di Gauss-Jordan (segue).

EQUAZIONI MAL CONDIZIONATE

Un matrice singolare ha determinante nullo, e il corrispondente sistema di equazioni o non ha soluzione o ne ha infinite. La matrice di un sistema mal condizionato, invece, è "quasi" singolare, produce risultati errati o imprecisi. Una piccola variazione nei dati di ingresso può provocare grosse variazioni nei risultati. Un esempio di matrice mal condizionata a due dimensioni è quella che si ottiene dalle equazioni di due rette quasi parallele: la soluzione è data dalle coordinate del punto di intersezione, ma tanto più le rette sono vicine alla condizione di parallelismo, tanto più difficile è determinare l'esatto punto di intersezione.

Esistono diversi test per sistemi mal condizionati; uno di questi consiste nel confrontare i valori della matrice inversa con quelli dell'originale. Se c'è una differenza di molti ordini di grandezza è probabile che il sistema sia mal condizionato. Un altro consiste nel confrontare l'inversa dell'inversa della matrice dei coefficienti con l'originale; dovrebbero essere identiche. Questo test controlla sia il procedimento di inversione che l'aritmetica del computer. Un ultimo test confronta tra di loro i valori che si trovano lungo la diagonale principale, che non dovrebbero essere molto diversi tra di loro.

La matrice di Hilbert è un esempio di matrice mal condizionata; è una matrice simmetrica che ha un 1 in alto a sinistra, mentre gli altri elementi diventano sempre più piccoli proseguendo nelle righe e nelle colonne, secondo lo schema seguente:

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & \cdots\cdots & 1/n \\ 1/2 & 1/3 & 1/4 & 1/5 & & 1/(n+1) \\ 1/3 & 1/4 & 1/5 & 1/6 & \cdots & 1/(n+2) \\ 1/4 & 1/5 & 1/6 & 1/7 & & 1/(n+3) \\ \cdots & & & & & \cdots \\ 1/n & \cdots & \cdots & \cdots & \cdots & 1/(2n-1) \end{bmatrix}$$

La matrice di Hilbert può essere usata per generare un sistema di equazioni mal condizionate, come:

$$x_1 + \frac{x_2}{2} + \frac{x_3}{3} + \frac{x_4}{4} + \frac{x_5}{5} = 1 + 1/2 + 1/3 + 1/4 + 1/5$$

$$\frac{x_1}{3} + \frac{x_2}{4} + \frac{x_3}{5} + \frac{x_4}{6} + \frac{x_5}{7} = 1/3 + 1/4 + 1/5 + 1/6 + 1/7$$

$$\frac{x_1}{2} + \frac{x_2}{3} + \frac{x_3}{4} + \frac{x_4}{5} + \frac{x_5}{6} = 1/2 + 1/3 + 1/4 + 1/5 + 1/6$$

$$\frac{x_1}{4} + \frac{x_2}{5} + \frac{x_3}{6} + \frac{x_4}{7} + \frac{x_5}{8} = 1/4 + 1/5 + 1/6 + 1/7 + 1/8$$

$$\frac{x_1}{5} + \frac{x_2}{6} + \frac{x_3}{7} + \frac{x_4}{8} + \frac{x_5}{9} = 1/5 + 1/6 + 1/7 + 1/8 + 1/9$$

Notiamo, prima di tutto che le frazioni 1/3, 1/6 e 1/7 non si possono calcolare esattamente, per cui ci saranno subito errori di arrotondamento. Inoltre, l'inversa della matrice dei coefficienti è:

$$\begin{bmatrix} 25 & -300 & 1050 & -1400 & 630 \\ -300 & 4800 & -18900 & 26880 & -12600 \\ 1050 & -18900 & 79380 & -117600 & 56700 \\ -1400 & 26880 & -117600 & 179200 & -88200 \\ 630 & -12600 & 56700 & -88200 & 44100 \end{bmatrix}$$

Alcuni elementi sono diversi ordini di grandezza maggiori di quelli della matrice originale inoltre il determinante è quasi zero.

Poichè ogni elemento del vettore dei termini noti è la somma degli elementi della matrice nella riga corrispondente, la soluzione esatta è:

[1 1 1 1 1]

Ma poichè la matrice è mal condizionata, la soluzione calcolata risulta essere all'incirca:

[1.0001 0.99816 1.00778 0.9884 1.0056]

Studiamo ora un programma che usa le matrici di Hilbert per studiare sistemi mal condizionati.

PROGRAMMA PASCAL: SOLUZIONE DELLE MATRICI DI HILBERT

Il programma della figura 4.9 genera una matrice di Hilbert e un vettore dei termini noti corrispondenti ad una soluzione:

[1 1 1 1]

Esecuzione del programma

Scrivete il programma ed eseguitelo; il programma gira senza il vostro intervento. Comincia con le due equazioni:

$$\begin{bmatrix} 1 & 1/2 \\ 1/2 & 1/3 \end{bmatrix} \quad \begin{bmatrix} 3/2 \\ 5/6 \end{bmatrix}$$

che vengono risolte con la prima versione della procedura di Gauss-Jordan. Questa matrice non è mal condizionata, e quindi apparentemente non esistono problemi. Il vettore soluzione è [1 1]. Il programma continua poi con tre, quattro, cinque, sei e sette equazioni.

Se il vostro compilatore Pascal esegue le operazioni in floating-point con la normale precisione di 32 bit, comincerete a notare gli errori di arrotondamento con quattro equazioni. Con sei equazioni avrete la precisione di due o tre cifre significative, sette equazioni daranno un risultato privo di significato.

Se invece avete una precisione di 64 bit, la situazione è molto diversa: la soluzione di un sistema di 17 equazioni avrà una precisione di almeno sette cifre. Dunque questo programma può essere usato per provare la precisione della vostra aritmetica in floating-point.

```
PROGRAM SOLVIT(OUTPUT);
(* 8 Feb 81 *)
(* matrice inversa di Hilbert n-per-n *)
(* la soluzione è 1 1 1 1 1 *)

(* Programma Pascal che esegue la soluzione di un sistema *)
(* per mezzo della eliminazione di Gauss-Jordan *)

CONST
  MAXR = 7;
  MAXC = 7;

TYPE
  ARY    = ARRAY[1..MAXR] OF REAL;
  ARYS   = ARRAY[1..MAXC] OF REAL;
  ARY2S  = ARRAY[1..MAXR, 1..MAXC] OF REAL;
VAR
  Y        : ARYS;
  COEF     : ARYS;
  A, B     : ARY2S;
  N, M, I, J: INTEGER;
  ERROR    : BOOLEAN;
```

Figura 4.9 — Risoluzione di un sistema di equazioni mal condizionate.

```
PROCEDURE GET_DATA(VAR A : ARY2S;
                   VAR Y : ARYS;
                   VAR N, M : INTEGER);
(* carica la matrice di Hilbert n-per-n *)
VAR
  I, J : INTEGER;

BEGIN
  FOR I := 1 TO N DO
    BEGIN
      A[N,I] := 1.0/(N + I - 1);
      A[I,N] := A[N,I]
    END;
  A[N,N] := 1.0/(2 * N - 1);
  FOR I := 1 TO N DO
    BEGIN
      Y[I] := 0.0;
      FOR J := 1 TO N DO
        Y[I] := Y[I] + A[I,J]
    END;

   WRITELN;
   IF N < 7 THEN
     BEGIN
       FOR I:= 1 TO N DO
         BEGIN
           FOR J:= 1 TO M DO
             WRITE( A[I,J] :7:5, ' ');
           WRITELN( ' : ', Y[I] :7:5)
         END;
       WRITELN
     END (* if n < 7 *)
  END (* procedure get_data *);
```

Figura 4.9 — Risoluzione di un sistema di equazioni mal condizionate (segue).

```
PROCEDURE WRITE_DATA;
(* stampa i risultati *)

VAR
   I : INTEGER;
BEGIN
   FOR I := 1 TO M DO
      WRITE( COEF[I] :9:5);
   WRITELN;
END (* write_data *);

(* PROCEDURE gaussj
   (VAR a     : ary2s;
    y         : arys;
    VAR coef  : arys;
    ncol      : integer;
    VAR error : boolean);
extern; *)
(*$F GAUSSJ.PAS *)
BEGIN (* programma principale *)
   A[1,1] := 1.0;
   N := 2;
   M := N;
   REPEAT
      GET_DATA (A, Y, N, M);
      FOR I := 1 TO N DO
         FOR J := 1 TO N DO
            B[I,J] := A[I,J] (* carica la matrice di lavoro *);
      GAUSSJ (B, Y, COEF, N, ERROR);
      IF NOT ERROR THEN WRITE_DATA;
      N := N + 1;
      M := N
   UNTIL N > MAXR
END.
```

Figura 4.9 — Risoluzione di un sistema di equazioni mal condizionate (segue).

Nel prossimo paragrafo vedremo un'altra variante della procedura di Gauss-Jordan, per un sistema con un numero di equazioni maggiore del numero di incognite, e presenteremo un programma per determinare la miglior curva interpolante.

MIGLIOR CURVA INTERPOLANTE PER UN SISTEMA DI EQUAZIONI

Il programma precedente produce un esatta interpolazione per un sistema con un numero di equazioni uguale a quello delle incognite. Se ci sono più incognite che equazioni, non esiste un'unica soluzione, e si ha una matrice con più colonne che righe.

C'è un altro caso da considerare: supponiamo di voler determinare il valore di m incognite con metodi sperimentali. Se si eseguono m misure indipendenti si può trovare una soluzione esatta; se invece il numero delle misure indipendenti n è maggiore del numero di incognite m è possibile determinare la miglior curva interpolante.

Consideriamo le tre equazioni:

$$\begin{aligned} x + y &= 3 \\ x \quad\;\; &= 1 \\ \quad\;\; y &= 1 \end{aligned}$$

che rappresentano tre rette che si incontrano in tre punti diversi; ogni coppia di rette definisce un punto nel piano x-y, e questi punti sono i vertici di un triangolo rettangolo, nelle posizioni (1,1), (2,1) e (1,2) come si vede nella figura 4.10. La scelta migliore per l'"intersezione" delle tre rette è il punto che si trova a un terzo del segmento di perpendicolare che va dalla base al vertice opposto:

$$x + 1.3333,\ y = 1.3333$$

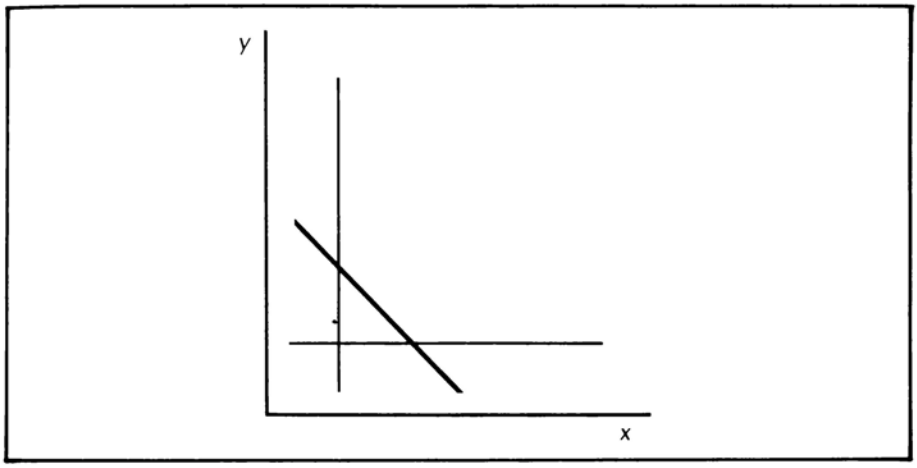


Figura 4.10 — Miglior interpolante bidimensionale.

Vediamo ora il programma che trova questa soluzione.

PROGRAMMA PASCAL: MIGLIOR CURVA INTERPOLANTE

Riscrivete il programma della figura 4.5; inserite la procedure SQUARE per convertire la matrice rettangolare dei coefficienti e il vettore dei termini noti nella matrice quadrata e nel vettore richiesti dalla procedure GAUSSJ, che per altro non compare in questo programma ed è richiamata tramite la direttiva INCLUDE.

L'algoritmo implementato da questo programma è quello dell'interpolazione con i minimi quadrati. Poichè questo argomento è studiato a fondo nel prossimo capitolo, qui non diremo altro.

```
PROGRAM SOLVGJ(INPUT,.OUTPUT);
(* 12 Feb 81 *)
(* Programma Pascal che esegue la soluzione di un sistema *)
(* per mezzo della eliminazione di Gauss-Jordan *)
(* ci possono essere più equazioni delle incognite *)

CONST
  MAXR = 8;
  MAXC = 8;
TYPE
  ARY   = ARRAY[1..MAXR] OF REAL;
  ARYS  = ARRAY[1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXR, 1..MAXC] OF REAL;
  ARY2  = ARY2S (* per SQUARE *);

VAR
  Y        : ARY;
  COEF, YY : ARYS;
  A, B     : ARY2S;
  N, M, I, J : INTEGER;
  ERROR    : BOOLEAN;

PROCEDURE GET_DATA(VAR A : ARY2S;
                   VAR Y : ARY;
                   VAR N, M : INTEGER);
```

Figura 4.11 — Miglior interpolante per un sistema di equazioni lineari.

```
(* riceve i valori per n e le matrici a, y *)

VAR
   I, J : INTEGER;

BEGIN
   WRITELN;
   REPEAT
      WRITE ('Quante incognite?');
      READLN( M)
   UNTIL M < MAXC;
   IF M > 1 THEN
      BEGIN
         REPEAT
            WRITE ('Quante equazioni?');
            READLN( N)
         UNTIL N >= M;
         FOR I := 1 TO N DO
            BEGIN
               WRITELN ('Equazione', 1:3);
               FOR J := 1 TO M DO
                  BEGIN
                     WRITE( J:3, ': ');
                     READ( A[I,J])
                  END;
               WRITE( ', C: ');
               READLN (Y[I])  (* linea bianca *)
            END (* ciclo i *);
         WRITELN;
         FOR I:= 1 TO N DO
            BEGIN
```

Figura 4.11 — Miglior interpolante per un sistema di equazioni lineari. (segue)

```
                FOR J:= 1 TO M DO
                   WRITE( A[I,J] :7:4, ' ');
                WRITELN( ' : ', Y[I] :7:4)
             END;
          WRITELN
       END (* if n > 1 *)
END (* procedure get_data *);

PROCEDURE WRITE_DATA;
(* stampa i risultati *)
VAR
   I : INTEGER;

BEGIN
   FOR I := 1 TO M DO WRITE( COEF[I] :9:5);
   WRITELN
END (* write_data *);

(* PROCEDURE square(x : ary2;
                     y : ary;
                VAR a : ary2s;
                VAR g : arys;
           nrow,ncol : integer);
extern; *)
(*$F SQUARE.PAS *)

(* PROCEDURE gaussj
   (VAR b     : ary2s;
    y         : arys;
    VAR coef  : arys;
    ncol      : integer;
    VAR error : boolean);
extern; *)
(*$F GAUSSJ.PAS *)
```

Figura 4.11 — Miglior interpolante per un sistema di equazioni lineari (segue).

```
BEGIN (* programma principale *)
  WRITELN;
  WRITELN ('La migliore approssimazione di un sistema di equazioni');
  WRITELN ('Con Gauss-Jordan');
  REPEAT
    GET_DATA (A, Y, N, M);
    IF M > 1 THEN
      BEGIN
        SQUARE(A, Y, B, YY, N, M);
        GAUSSJ (B, YY, COEF, M, ERROR);
        IF NOT ERROR THEN WRITE_DATA
      END
  UNTIL M < 2
END.
```

Figura 4.11 — Miglior interpolante per un sistema di equazioni lineari (segue).

Esecuzione del programma

Compilate il programma ed eseguitelo. Prima viene richiesto il numero delle incognite, poi quello delle equazioni; se sono uguali viene calcolata e restituita la soluzione esatta, altrimenti la miglior curva interpolante. Le tre equazioni del paragrafo precedente avevano due incognite. La matrice e il vettore corrispondenti sono:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}$$

Verificate che il risultato sia:

$$x = 1.3333, \quad y = 1.3333$$

Consideriamo ora il circuito presentato all'inizio del capitolo, per il quale abbiamo già scritto e risolto il sistema delle tre equazioni delle correnti per le tre maglie. Supponiamo ora che le tensioni dei generatori siano determinate sperimentalmente; se,

per il generatore di sinistra si trova un valore di 19 volt, e per quello di destra di −5.1 volt, le tre equazioni diventano:

$$\begin{bmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \end{bmatrix} \qquad \begin{bmatrix} 19 \\ -5.1 \\ 0 \end{bmatrix}$$

Inoltre supponiamo che la caduta di tensione misurata sulla resistenza di 1 ohm, che si trova sul ramo orizzontale, sia di 1.1 volt. La corrente che attraversa questa resistenza è data dalla differenza tra I_2 e I_3; questo ci consente di scrivere un'altra equazione indipendente, e cioè un'altra riga nella nostra matrice. Le quattro equazioni sono:

$$\begin{bmatrix} 13 & -8 & -3 \\ -8 & 10 & -1 \\ -3 & -1 & 11 \\ 0 & 1 & -1 \end{bmatrix} \qquad \begin{bmatrix} 19 \\ -5.1 \\ 0 \\ 1.1 \end{bmatrix}$$

Inserite queste quattro equazioni, con le loro tre incognite, nel nuovo programma, e confrontate la curva miglior interpolante:

$$I_1 = 2.8 \text{ amps}, \qquad I_2 = 1.8 \text{ amps}, \qquad I_3 = 0.94 \text{ amps}$$

con la soluzione originale. Inserite zero come numero di equazioni per uscire dal programma.

Ora studieremo un metodo — e il corrispondente programma Pascal — per risolvere sistemi di equazioni con coefficienti complessi (cioè numeri che contengono una parte immaginaria), e, come esempio, prenderemo un circuito elettrico un po' più complicato del primo.

EQUAZIONI CON COEFFICIENTI COMPLESSI

Nello studio dei circuiti elettrici compaiono sistemi di equazioni con coefficienti complessi. I numeri complessi non sono previsti come "tipo" dai compilatori standard, ma possono essere definiti dall'utente; Jensen e Wirth (1) hanno trovato un metodo per generare numeri complessi usando il tipo record. È però più facile risolvere un sistema di n equazioni complesse trasformandole in 2n equazioni reali, e risolvendo il sistema così ottenuto con uno dei metodi già visti.

(1) Jensen e Wirth, Pascal Manuale standard del linguaggio.

Esempio: un circuito elettrico in corrente alternata

Consideriamo il circuito elettrico della figura 4.12; è più complicato di quello visto finora perchè contiene un generatore di tensione alternata, un'induttanza e un condensatore. L'impedenza di una resistenza è semplicemente R, mentre quella di un'induttanza e di un condensatore è funzione della frequenza, e precisamente l'impedenza di un'induttanza è $j\omega$ L, dove j è il numero immaginario che si ottiene dalla radice di -1, ω è la frequenza del generatore in radianti al secondo, L è il valore dell'induttanza in Henry. L'impedenza di un condensatore è $-j/\omega C$, dove C è il valore della capacità in Farad.

Nella figura 4.12 i valori delle impedenze sono indicate vicino ai vari componenti; la tensione del generatore è 10 volt con una frequenza ω. L'induttanza ha un'impedenza di j5 ohm, e il condensatore di $-j4$ ohm.

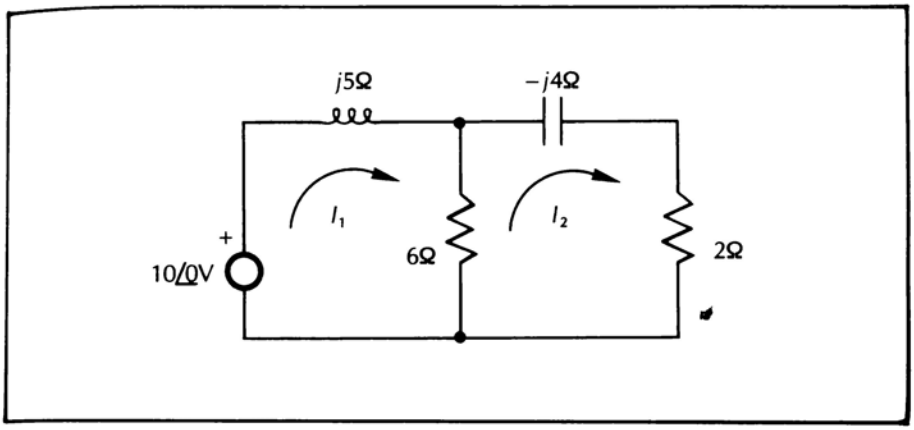


Figura 4.12 — Circuito con generatore di tensione continua.

Possiamo trovare le correnti di questo circuito con le leggi di Kirchoff; muovendosi in ogni maglia in senso orario, si trova:

$$(6 + j5)\, I_1 - 6\, I_2 - 10 = 0 \quad \text{(maglia di sinistra)}$$
$$-6\, I_1 + (8 - j4)\, I_2 = 0 \quad \text{(maglia di destra)}$$

La matrice e il vettore corrispondenti sono:

$$\begin{bmatrix} (6 + j5) & (-6 + j0) \\ (-6 + j0) & (8 - j4) \end{bmatrix} \qquad \begin{bmatrix} (10 + j0) \\ (0 + j0) \end{bmatrix}$$

Queste due equazioni non possono essere risolte direttamente con i programmi che abbiamo visto finora, perchè contengono coefficienti complessi.

Consideriamo ora una rappresentazione generale delle equazioni alle maglie: sia le correnti che le impedenze possono essere espresse come numeri complessi.

Possiamo dunque scrivere:

$$(AR_{11} + jAI_{11})\,(IR_1 + jII_1) + (AR_{21} + jAI_{21})\,(IR_2 + jII_2) = (VR_1 + jVI_1)$$

$$(AR_{21} + jAI_{21})\,(IR_1 + jII_1) + (AR_{22} + jAI_{22})\,(IR_2 + jII_2) = (VR_2 + jVI_2)$$

dove:

AR_{kl} = parte reale del coefficiente (impedenza) k,l
AI_{kl} = parte immaginaria del coefficiente k,l
IR_l = parte reale della corrente l
II_l = parte immaginaria della corrente l
VR_k = parte reale della tensione nell'equazione k
VI_k = parte immaginaria della tensione nell'equazione k

Eseguendo la moltiplicazione nel primo membro delle equazioni precedenti si ottengono risultati in cui, alternativamente, compare l'operatore complesso j.

$$(AR_{11}\,IR_1 - AI_{11}\,II_1) + j(AR_{11}\,II_1 + AI_{11}\,IR_1) +$$

Per essere soddisfatta l'uguaglianza dell'espressione a sinistra dell'uguale con quella a destra, la parte reale a sinistra deve essere uguale a quella di destra, e i coefficienti della parte immaginaria a sinistra devono essere uguali a quelli di destra. Queste condizioni portano a scrivere un sistema di 2n equazioni in cui compaiono solo coefficienti reali.

La prima equazione è:

$$AR_{11} - AI_{11} + AR_{12} - AI_{12} = VR_1$$

Notate che vi compaiono i complessi coniugati dei coefficienti originali, cioè i coefficienti originali compaiono nell'ordine, ma con i segni cambiati.

La seconda equazione è:

$$AI_{11} + AR_{11} + AI_{12} + AR_{12} = VI_1$$

Anche questa contiene i coefficienti dell'originale, ma le parti reali e immaginarie sono scambiate fra di loro, e sono restati i segni originali.

Il nuovo sistema di equazioni completo dà origine alle seguenti matrici:

$$\begin{bmatrix} AR_{11} & -AI_{11} & AR_{12} & -AI_{12} \\ AI_{11} & AR_{11} & AI_{12} & AR_{12} \\ AR_{21} & -AI_{21} & AR_{22} & -AI_{22} \\ AI_{21} & AR_{21} & AI_{22} & AR_{22} \end{bmatrix} \qquad \begin{bmatrix} VR_1 \\ VI_1 \\ VR_2 \\ VI_2 \end{bmatrix}$$

Sostituendo i valori della figura 4.12 otteniamo:

$$\begin{bmatrix} 6 & -5 & -6 & 0 \\ 5 & 6 & 0 & -6 \\ -6 & 0 & 8 & 4 \\ 0 & -6 & -4 & 8 \end{bmatrix} \qquad \begin{bmatrix} 10 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Ogni coefficiente originale compare due volte nella nuova matrice. Il vettore soluzione può essere determinato facilmente con i metodi studiati in questo capitolo, ed è:

$$[1.5 \quad -2.0 \quad 1.5 \quad -0.75]$$

che corrisponde alle correnti:

$$I_1 = 1.5 - j2 \text{ amps} = 2.5 \angle -53^\circ$$
$$I_2 = 1.5 - j0.75 \text{ amps} = 1.67 \angle -27^\circ$$

Possiamo verificare l'esattezza di questi risultati calcolando le cadute di tensione su ogni elemento del circuito. Per esempio, se supponiamo che il nodo in basso sia a zero volt, la tensione sul nodo superiore è uguale alla caduta di tensione sulla resistenza di 6 ohm:

$$V = 6(I_1 - I_2) = 6(-j1.25) = -j7.5 \text{ volt}$$

Analogamente la caduta di tensione sull'induttanza è:

$$V = j5(I_1) = 10 + j7.5 \text{ volt}$$

La somma delle cadute di tensione nella maglia di sinistra è:

$$(-j7.5) + (10 + j7.5) - (10) = 0$$

La stessa verifica si può fare sulla maglia di destra.

Studiamo ora un programma per risolvere sistemi di equazioni con coefficienti complessi.

PROGRAMMA PASCAL: SISTEMI DI EQUAZIONI CON COEFFICIENTI COMPLESSI

Il programma della figura 4.13 facilita la soluzione di sistemi di equazioni con coefficienti complessi. Ogni coefficiente delle n equazioni originali viene inserito una sola volta; il programma sistema poi i dati in una matrice 2n x 2n e in un vettore dei termini noti di lunghezza 2n. Si possono risolvere sistemi con al massimo 4 equazioni; questo limite si può cambiare cambiando i valori alle variabili NROW e NCOL, che contengono un numero pari al doppio del numero massimo di equazioni. Uno qualsiasi dei metodi sviluppati in questo capitolo va bene per trovare la soluzione; noi abbiamo scelto quello di Gauss-Jordan, per cui è meglio iniziare con una copia del programma di Gauss-Jordan di figura 4.5.

```
PROGRAM SOLVEC(INPUT, OUTPUT);
(* 8 Mar 81 )*
(* Programma Pascal che esegue la soluzione di un sistema *)
(* per coefficienti complessi *)
(* per mezzo della eliminazione di Gauss-Jordan *)


CONST
  MAXR = 8;
  MAXC = 8;
TYPE
  ARY   = ARRAY[1..MAXR] OF REAL;
  ARYS  = ARRAY[1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXR, 1..MAXC] OF REAL;
  ARYC2 = ARRAY[1..MAXR, 1..MAXC, 1..2] OF REAL;
  ARYC  = ARRAY[1..MAXR, 1..2] OF REAL;

VAR
  Y        : ARYS;
  COEF     : ARYS;
  A, B     : ARY2S;
  N, M, I, J : INTEGER;
  ERROR    : BOOLEAN;
```

Figura 4.13 — Risoluzione di sistemi di equazioni con coefficienti complessi.

```
PROCEDURE GET_DATA(VAR A : ARY2S;
                   VAR Y : ARYS;
                   VAR N, M : INTEGER);

(* riceve i valori complessi di n e delle matrici a, y *)

VAR
  C : ARYC2;
  V : ARYC;
  I, J, K, L : INTEGER;

PROCEDURE SHOW;
(* stampa i nuovi dati *)

VAR
  I, J, K : INTEGER;

BEGIN (* show *)
  WRITELN;

  FOR I:= 1 TO N DO
    BEGIN
      FOR J:= 1 TO M DO
        FOR K := 1 TO 2 DO
          WRITE( C[I,J,K] :7:4, ' ');
      WRITELN( ' : ', V[I,1] :7:4, ' : ', V[I,2] :7:4)
    END;
  N := 2 * N;
  M := N;
  WRITELN;
  FOR I:= 1 TO N DO
    BEGIN
```

Figura 4.13 — Risoluzione di sistemi di equazioni con coefficienti complessi (segue).

```
        FOR J:= 1 TO M DO
           WRITE( A[I,J] :7:4, ' ');
        WRITELN( ' : ', Y[I] :9:5)
     END;
   WRITELN
END (* show *);
BEGIN (* procedure get_data *)
   WRITELN;
   REPEAT
      WRITE ('Quante equazioni?');
      READLN( N);
      M := N
   UNTIL N < MAXR;
   IF N > 1 THEN
      BEGIN
         FOR I := 1 TO N DO
            BEGIN
               WRITELN ('Equazione', 1:3);
               K := 0;
               L := 2 * I - 1;

              FOR J := 1 TO N DO
                 BEGIN
                    K := K + 1;
                    WRITE ('Reale', J:3, ':');
                    READ( C[I,J, 1]) (* parte reale *);
                    A[L,K] := C[I,J, 1];
                    A[L+1,K+1] := C[I,J, 1];
                    K := K + 1;
                    WRITE ('Imm', J:3, ':');
                    READ( C[I,J, 2]) (* parte immaginaria *);
                    A[L,K] := -C[I,J, 2];
                    A[L+1,K-1] := C[I,J, 2]
```

Figura 4.13 — Risoluzione di sistemi di equazioni con coefficienti complessi (segue).

```
            END (* ciclo j *);
          WRITE ('const. Reale');
          READ (V[I,1]) (* costante reale *);
          Y[L] := V[I,1];
          WRITE ( 'Const imm.: ');
          READLN (V[I,2]) (* costante immaginaria *)
          Y[L+1] := V[I,2]
        END (* ciclo i *);
      SHOW (* dati originali *)
    END (* if n > 1 *)
END (* procedure get_data *);

PROCEDURE WRITE_DATA;

(* stampa i risultati *)

VAR
  I, J : INTEGER;
  RE, IM : REAL;

FUNCTION MAG(X, Y : REAL): REAL;
(* magnitudine polare *)
BEGIN
  MAG := SQRT( SQR(X) + SQR(Y))
END (* function mag *);

FUNCTION ATAN(X, Y : REAL): REAL;
(* arctan in gradi *)
CONST
  PI180 = 57.2957795;
```

Figura 4.13 — Risoluzione di sistemi di equazioni con coefficienti complessi (segue).

```
VAR
  A : REAL;

BEGIN (* atan *)
  IF X = 0.0 THEN
    IF Y = 0.0 THEN ATAN := 0.0
    ELSE ATAN := 90.0
  ELSE (* x e y <> 0 *)
    IF Y = 0.0 THEN ATAN := 0.0
    ELSE (* x and y <> 0 *)
      BEGIN
        A := ARCTAN(ABS(Y/X)) * PI180;
        IF X > 0.0 THEN
          IF Y > 0.0 THEN ATAN := A (* x, y > 0 *)
          ELSE ATAN := -A (* x > 0, y < 0 *)
        ELSE (* x < 0 *)
          IF Y > 0.0 THEN ATAN := 180.0 - A
          (* x < 0, y > 0 *)
          ELSE ATAN := 180.0 + A (* x, y < 0 *)
      END (* else *)
END (* function atan *);

BEGIN
  WRITELN
    ('Reale Immaginaria Magnitudine Angolo');
  FOR I := 1 TO M DIV 2 DO
    BEGIN
      J := 2 * I - 1;
      RE := COEF[J];
      IM := COEF[J+1];
      WRITELN( RE :11:5, IM :11:5,
        MAG(RE, IM) :11:5,
        ATAN(RE, IM) :11:5)
```

Figura 4.13 — Risoluzione di sistemi di equazioni con coefficienti complessi (segue).

```
      END (* for *);
   WRITELN
END (* write_data *);

(* PROCEDURE gaussj
   (VAR b     : ary2s;
    y         : arys;
    VAR coef  : arys;
    ncol      : integer;
    VAR error : boolean);
extern; *)
(*$F GAUSSJ.PAS *)


BEGIN (* main program *)
   WRITELN;
   WRITELN
      ('Soluzione di un sistema con coefficienti complessi');
   WRITELN ('Per mezzo della eliminazione di Gauss-Jordan');
   REPEAT
      GET_DATA (A, Y, N, M);

      IF N > 1 THEN
         BEGIN
            FOR I := 1 TO N DO
               FOR J := 1 TO N DO
                  B[I,J] := A[I,J] (* carica della matrice di lavoro *);
            GAUSSJ (B, Y, COEF, N, ERROR);
            IF NOT ERROR THEN WRITE_DATA
         END
   UNTIL N < 2
END.
```

Figura 4.13 — Risoluzione di sistemi di equazioni con coefficienti complessi (segue).

Esecuzione del programma

Battete il programma e usatelo per risolvere il circuito della figura 4.12. Inserite 2 come numero delle equazioni, e poi i coefficienti delle equazioni nell'ordine corretto, prima la parte reale e poi quella immaginaria; nello stesso modo inserite il vettore dei termini noti. I valori da inserire nel nostro caso sono:

```
 6, 5,  -6,   0, 10,  0   (equazione 1)
-6, 0,   8,  -4,  0,  0   (equazione 2)
```

I dati di ingresso sono stampati, seguiti dalla nuova matrice 2n x 2n; poi viene data la soluzione, sia in forma rettangolare che in forma polare.

L'ampiezza della rappresentazione polare viene calcolata nella funzione MAG, come risultato della radice quadrata della somma dei quadrati delle componenti rettangolari. L'angolo di sfasamento viene calcolato dalla funzione ATAN; questa funzione sembra notevolmente più complicata del necessario. Il problema è che le funzioni ARCTAN non hanno una versione standard: come abbiamo visto nel capitolo 1, alcune restituiscono un valore tra 0° e 180°, altre tra –90° e +90°. La funzione ATAN calcola l'arcotangente nel primo quadrante; solo in seguito l'angolo viene convertito nel quadrante esatto.

Fino a questo punto abbiamo visto diversi metodi per risolvere piccoli sistemi di equazioni; l'ultimo argomento che affronteremo in questo capitolo è un metodo iterativo adatto a risolvere grossi problemi.

IL METODO ITERATIVO DI GAUSS-SEIDEL

I metodi di eliminazione di Gauss e di Gauss-Jordan che abbiamo visto finora non si prestano alla risoluzione di grossi sistemi di equazioni, in quanto il numero di moltiplicazioni e sottrazioni cresce notevolmente al crescere del numero di equazioni, e il conseguente errore di arrotondamento può produrre risultati privi di significato.

Il metodo di Gauss-Seidel determina la soluzione di un sistema di equazioni con una tecnica iterativa, in cui un primo risultato approssimato viene ripetutamente aggiustato, fino a quando diventa accettabilmente vicino alla soluzione. Poichè ogni approssimazione dipende solo da quella precedente, gli errori di arrotondamento non si accumulano; inoltre non è necessario che le equazioni siano lineari.

Consideriamo le tre equazioni delle maglie della figura 4.1:

$$
\begin{aligned}
13I_1 - 8I_2 - 3I_3 &= 20 \\
-8I_1 + 10I_2 - I_3 &= -5 \\
-3I_1 - I_2 + 11I_3 &= 0
\end{aligned}
$$

Ricavando I_1 dalla prima equazione si ottiene:

$$I_1 = \frac{20 + 8I_2 + 3I_3}{13}$$

dove I_1 è espresso in funzione delle altre due incognite. Se prendiamo zero come prima approssimazione per I_2 e I_3, otteniamo per I_1 il valore di 1.54. Ora possiamo risolvere la seconda equazione rispetto alla seconda incognita:

$$I_2 = \frac{8I_1 + I_3 - 5}{10}$$

Sostituendo 1.54 per I_1 e 0 per I_3 otteniamo un valore di 0.73 per I_2. Analogamente si risolve la terza equazione rispetto alla terza incognita:

$$I_3 = \frac{3I_1 + I_2}{11}$$

Sostituendo 1.54 per I_1 e 0.73 per I_2 otteniamo 0.486 per I_3. Il procedimento viene poi ripetuto, usando i valori 0.73 per I_2 e 0.486 per I_3 e ricavando un valore con una migliore approssimazione per I_1. Dopo circa 20 iterazioni i valori hanno una precisione di tre cifre significative. La seguente tabella dà nell'ordine alcuni di questi valori:

I_1	I_2	I_3
0	0	0
1.54	0.73	0.86
2.1	1.23	0.685
2.45	1.53	0.808
2.67	1.71	0.883

Nel metodo di Gauss-Seidel si possono presentare diversi problemi: prima di tutto il procedimento può non convergere, cioè i valori che man mano si trovano possono allontanarsi sempre più dalla soluzione esatta. Consideriamo ad esempio le tre equazioni precedenti: se le scriviamo nell'ordine inverso otteniamo:

$$I_1 = (11I_3 - I_2) / 3$$
$$I_2 = (13I_1 - 3I_3 - 20) / 13$$
$$I_3 = 10I_2 - 8I_1$$

Scegliamo come prima approssimazione lo zero, come nel caso precedente. La successione di valori che si trova è chiaramente divergente:

I_1	I_2	I_3
0	0	0
0	−1.5	−15
−57	−54	−92
−318	−299	−440

Il motivo è che i valori maggiori non si trovano sulla diagonale principale; basta cambiare le righe per mettere gli elementi di valore maggiore nelle posizioni del pivot.

Studiamo ora un'implementazione Pascal del metodo di Gauss-Seidel, prestando particolare attenzione ad un paio di caratteristiche interessanti:

. la differenza tra criteri relativi e assoluti nelle decisioni **IF...THEN**
. il significato e l'uso del "rilassamento per punti"

PROGRAMMA PASCAL: IL METODO DI GAUSS-SEIDEL

Stranamente, la scelta della prima approssimazione non è molto importante, mentre un nuovo problema da considerare è il criterio di convergenza. Il programma della figura 4.14 può essere usato per provare il metodo di Gauss-Seidel; gran parte del programma può essere copiata da quello di eliminazione di Gauss della figura 4.3.

La routine di scambio dei pivot usata con il metodo di eliminazione di Gauss è inserita anche in questo programma con lo stesso scopo, per scambiare fra di loro le righe in modo di avere sulla diagonale principale l'elemento maggiore di ogni colonna.

Per determinare la convergenza si usa un criterio assoluto:

```
IF ABS(NEXTC – COEF[J]) >TOL THEN...
```

In genere è meglio usare un criterio relativo:

```
IF ABS(1 – COEF[J] / NEXTC) >TOL THEN...
```

Ma in questo caso dobbiamo essere sicuri che NEXTC sia diverso da zero; possiamo usare la formula:

```
IF ABS(NEXTC – COEF[J]) >ABS(TOL*NEXTC) THEN...
```

```
PROGRAM GAUSID(INPUT, OUTPUT);
(* 23 Feb 81 *)
(* Programma Pascal che esegue *)
(* la soluzione di un sistema per mezzo di Gauss-Seidel *)
(* la procedura SEID è inclusa *)

CONST
  MAXR = 8;
  MAXC = 8;

TYPE
  ARY   = ARRAY[1..MAXR] OF REAL;
  ARYS  = ARRAY[1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXR, 1..MAXC] OF REAL;

VAR
  Y      : ARY;
  COEF   : ARYS;
  A      : ARY2S;
  N, M   : INTEGER;
  ERROR  : BOOLEAN;

PROCEDURE GET_DATA
  (VAR A     : ARY2S;
   VAR Y     : ARY;
   VAR N, M  : INTEGER);
(* riceve i valori di n e delle matrici a, y *)

VAR
  I, J : INTEGER;

BEGIN
  WRITELN;
```

Figura 4.14 — Risoluzione di equazioni lineari con il metodo di Gauss Seidel.

```
REPEAT
   WRITE('Quante equazioni?');
   READLN(N)
UNTIL N < MAXR;
M := N;
IF N > 1 THEN
   BEGIN
      FOR I := 1 TO N DO
         BEGIN
            WRITELN('Equazione', I:3);
            FOR J := 1 TO N DO
               BEGIN
                  WRITE(J: 3, ': ');
                  READ(A[I, J])
               END;
            WRITE(', C: ');
            READ(Y[I]);
            READLN (* linea bianca *)
         END;
      WRITELN;
      FOR I := 1 TO N DO
         BEGIN
            FOR J := 1 TO M DO
               WRITE(A[I, J]: 7: 4, ' ');
            WRITELN(' : ', Y[I]: 7: 4)
         END;
      WRITELN
   END (* if n > 1 *)
  ELSE IF N < 0 THEN N := - N;
  M := N
END (* procedure get_data *);
```

Figura 4.14 — Risoluzione di equazioni lineari con il metodo di Gauss Seidel (segue).

```
PROCEDURE WRITE_DATA;
(* stampa i risultati *)

VAR
  I : INTEGER;

BEGIN
  FOR I := 1 TO M DO
    WRITE(COEF[I]: 9: 5);
  WRITELN
END (* write_data *);

PROCEDURE SEID
          (A       : ARY2S;
           Y       : ARY;
      VAR COEF     : ARYS;
          NCOL     : INTEGER;
      VAR ERROR    : BOOLEAN);
(* soluzione della matrice per mezzo di Gauss-Seidel *)
(* 23 Feb 1981 *)
CONST
  TOL = 1.0E-4;
  MAX = 100;

VAR
  DONE : BOOLEAN;
  I, J, K, L, N : INTEGER;
  NEXTC, HOLD, SUM, LAMBDA, AB, BIG : REAL;

BEGIN
  REPEAT
    WRITE('Fattore di rilassamento?');
    READLN(LAMBDA)
```

Figura 4.14 — Risoluzione di equazioni lineari con il metodo di Gauss Seidel (segue).

```
UNTIL (LAMBDA < 2.0) AND (LAMBDA > 0.0);
ERROR := FALSE;
N := NCOL;
FOR I := 1 TO N - 1 DO
  BEGIN
    BIG := ABS(A[I, I]);
    L := I;
    FOR J := I + 1 TO N DO
      BEGIN
        (* ricerca il maggiore elemento *)
        AB := ABS(A[J, I]);
        IF AB > BIG THEN
          BEGIN
            BIG := AB;
            L := J
          END
      END (* ciclo j *);
    IF BIG = 0.0 THEN ERROR := TRUE
    ELSE
      BEGIN
        IF L <> I THEN
          BEGIN
            (* scambia le righe per porre *)
            (* l'elemento maggiore sulla diagonale *)
            FOR J := 1 TO N DO
              BEGIN
                HOLD := A[L, J];
                A[L, J] := A[I, J];
                A[I, J] := HOLD
              END;
            HOLD := Y[L];
            Y[L] := Y[I];
            Y[I] := HOLD
```

Figura 4.14 — Risoluzione di equazioni lineari con il metodo di Gauss Seidel (segue).

```
            END     (* if L <> i *)
        END         (* if big *)
    END;            (* ciclo i *)
IF A[N, N] = 0.0 THEN ERROR := TRUE
ELSE
    BEGIN
        FOR I := 1 TO N DO
            COEF[I] := 0.0 (* posizione iniziale *);
        I := 0;
        REPEAT
            I := I + 1;
            DONE := TRUE;
            FOR J := 1 TO N DO
                BEGIN
                    SUM := Y[J];
                    FOR K := 1 TO N DO
                        IF J <> K THEN
                            SUM := SUM - A[J, K] * COEF[K];
                    NEXTC := SUM/A[J, J];
                    IF ABS(NEXTC - COEF[J]) > TOL THEN
                        BEGIN
                            DONE := FALSE;
                            IF NEXTC * COEF[J] < 0.0 THEN
                                NEXTC := (COEF[J] + NEXTC) * 0.5
                        END;
                    COEF[J] :=
                        LAMBDA * NEXTC + (1.0 - LAMBDA)
                                * COEF[J];
                    WRITELN(I: 4, ', coef(', J, ') =', COEF[J])
                END (* ciclo J *)
        UNTIL DONE OR (I > MAX)
    END; (* IF a[n,n] = 0 *);
```

Figura 4.14 — Risoluzione di equazioni lineari con il metodo di Gauss Seidel (segue).

```
    IF I > MAX THEN ERROR := TRUE;
    IF ERROR THEN WRITELN ('ERRORE: Matrice singolare')
  END (* seid *);

  BEGIN              (* programma principale *)
    WRITELN;
    WRITELN('Soluzione simultanea per mezzo di Gauss-Seidel');
    REPEAT
      GET_DATA(A, Y, N, M);
      IF N > 1 THEN
        BEGIN
          SEID(A, Y, COEF, N, ERROR);
          IF NOT ERROR THEN WRITE_DATA
        END
    UNTIL N < 2
  END.
```

Figura 4.14 — Risoluzione di equazioni lineari con il metodo di Gauss Seidel (segue).

Qualche volta le approssimazioni successive differiscono eccessivamente; il programma della figura 4.14 attenua questo fenomeno. Se due successive approssimazioni hanno segno diverso la loro differenza è dimezzata.

Un'altra caratteristica di questo programma va sotto il nome di "rilassamento per punti": ogni valore è funzione della precedente iterazione, del valore calcolato, e di un fattore di rilassamento LAMBDA. Se COEF[J] è il valore precedente e NEXTC è il valore calcolato, il valore successivo diventa:

```
COEF[J] : = LAMBDA*NEXTC + (1.0 - LAMBDA) COEF[J] ;
```

Il valore di LAMBDA è compreso tra 0 e 2.

Esecuzione del programma

Scrivete il programma di figura 4.14 ed eseguitelo; vi chiederà il numero di equazioni e voi inserite 3. Poi inserite le tre equazioni del circuito elettrico, in un ordine qualsiasi, visto che è compresa la routine di scambio delle righe. A questo punto il programma vi chiederà il fattore di rilassamento rispondete 1.0. Il programma stamperà i valori di ogni successiva iterazione dopo il numero dell'iterazione stessa.

La convergenza si avrà dopo circa 20 iterazioni, e il programma vi chiederà ancora il numero di equazioni. Rispondete –3 per dire, con il segno –, che volete usare le stesse equazioni.

Potete eseguire il programma ripetutamente con gli stessi dati, cambiando il fattore di rilassamento. Nella seguente tabella vedete come il numero di iterazioni dipende dal fattore di rilassamento.

Lambda	Iterazioni
0.8	31
1.0	20
1.2	11
1.3	9
1.4	12
1.5	15
1.8	44

Ora inserite una matrice di Hilbert 2 x 2:

$$\begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 0.3333 \end{bmatrix} \qquad \begin{bmatrix} 1.5 \\ 0.83333 \end{bmatrix}$$

Troverete che il valore ottimale per il fattore di rilassamento è 1.4, circa lo stesso del sistema precedente. Per altri sistemi di equazioni sarà 1.0 o 0.8.

Come si vede, il metodo di Gauss-Seidel non è automatico come gli altri che abbiamo studiato, ma è utile per risolvere sistemi con un gran numero di equazioni o non lineari.

SOMMARIO

Abbiamo studiato un certo numero di metodi per risolvere sistemi di equazioni, ognuno adatto ad una situazione particolare. Abbiamo poi presentato i programmi che realizzano questi metodi sul calcolatore. Abbiamo anche studiato diversi casi particolari: vettore multiplo di termini noti, equazioni mal condizionate; ricerca della migliore interpolante per un sistema di equazioni indeterminato; equazioni con coefficienti complessi. Nei relativi programmi abbiamo visto poi diverse nuove e interessanti caratteristiche della programmazione in Pascal.

CAPITOLO 5

PROGRAMMA DI INTERPOLAZIONE DI UNA CURVA

INTRODUZIONE

In questo capitolo svilupperemo un programma che esegue l'interpolazione lineare di una curva coi minimi quadrati. In particolare studieremo un programma per trovare la retta che rappresenta nel modo migliore un insieme di dati x-y: il programma genera i dati, calcola l'equazione desiderata, stampa i risultati, produce un grafico dei dati e fornisce una misura della correlazione tra x e y. Nel capitolo 6 aggiungeremo una routine di ordinamento che consente il trattamento di dati sperimentali reali.

Il programma è lungo e complicato, ma noi non programmeremo tutte le parti insieme; useremo invece un approccio modulare, di tipo top-down, per cui cominceremo a scrivere solo una piccola parte del programma, che verrà immediatamente eseguita per provarla. Poi aggiungeremo un'altra parte e proveremo il nuovo programma. Con questa tecnica anche un programma relativamente complesso può essere sviluppato in modo logico e graduale, provando le singole parti man mano che si aggiungono; se si trova un errore sarà presumibilmente nell'ultima parte aggiunta.

Man mano che svilupperemo le diverse porzioni del programma, discuteremo diversi algoritmi e il modo per implementarli, e precisamente:

- l'uso di una funzione RANDOM e di un "fattore di falsificazione" per simulare dati sperimentali sparpagliati su di una retta
- una procedura per stampare grafici su un normale terminale o una stampante
- una procedura di interpolazione con i minimi quadrati che usa il calcolo differenziale per determinare la tangente e l'intercetta sull'asse y dei dati interpolati

- un metodo semplice ed elegante per calcolare il coefficiente di correlazione col nostro programma.

IL PROGRAMMA PRINCIPALE

Prima di tutto dobbiamo scrivere il programma principale con le routine di ingresso/uscita; il programma principale sarà il più breve possibile, e conterrà: il nome del programma, le istruzioni di dichiarazione e le chiamate alle procedure. Studieremo due versioni di questo programma, una che usa una funzione RANDOM di sistema, e una che simula questa funzione.

Prima versione: funzione RANDOM di sistema

Scrivete il programma Pascal della Figura 5.1 e dategli un nome significativo come:

CFIT1.PAS

Il programma principale inizia con la dichiarazione di INPUT e di OUTPUT, seguite immediatamente dalle variabili GLOBAL. Seguono due procedure, la GET-DATA che fornisce i dati di ingresso, e la WRITE-DATA che produce la stampa dei risultati. Le chiamate a questi programmi sono alla fine del programma principale; notate che l'ultima chiamata è seguita da un punto e virgola, che non è necessario anche se sintatticamente corretto; d'altra parte la presenza del punto e virgola in questo punto semplifica la programmazione della parte seguente.

La procedura di ingresso GET-DATA produce inizialmente un insieme di punti x-y con una funzione di generazione di numeri casuali; si potrebbe in un secondo tempo modificarla, in modo che legga i dati sperimentali da un file su disco.

Seconda versione: simulazione della funzione RANDOM

La procedura GET-DAT chiama una funzione RANDOM per generare una retta con punti sparpagliati in modo casuale; questa funzione non fa parte normalmente del linguaggio Pascal, ma potrebbe essere prevista dal vostro compilatore. In caso contrario, userete il programma della Figura 5.2, che comprende la funzione RANDOM, al posto di quelle della Figura 5.1.

```
PROGRAM CFIT1(INPUT,OUTPUT);

(* Programma Pascal che esegue una *)
(* interpolazione lineare coi minimi quadrati *)
(* 22 Ott 80 *)

CONST
  MAX = 20;

TYPE
  INDEX = 1..MAX;
  ARY   = ARRAY[INDEX] OF REAL;

VAR
  X, Y, Y_CALC : ARY;
  N      : INTEGER;
  DONE : BOOLEAN;
  A, B   : REAL;

PROCEDURE GET_DATA(VAR X, Y : ARY;
                   VAR N : INTEGER);

(* riceve i valori di n e delle matrici x, y *)
(* y è sparpagliato in modo random su una linea retta *)

CONST
  A = 2.0;
  B = 5.0;

VAR
  I, J     : INTEGER;
  FUDGE : REAL;
```

Figura 5.1 — Parte iniziale di un programma di interpolazione di una curva.

```
BEGIN
   WRITE ('Falsificazione?');
   READLN( FUDGE);
   IF FUDGE < 0.0 THEN DONE := TRUE
   ELSE
      BEGIN
         REPEAT
            WRITE ('Quanti punti?');
            READLN( N)
         UNTIL (N > 2) AND (N <= MAX);
         FOR I := 1 TO N DO
            BEGIN
               J := N + 1 - I;
               X[I] := J;
               Y[I] := (A + B*J)
                        *(1.0 + (2.0*RANDOM(0) - 1.0)
                                 *FUDGE)
            END (* ciclo FOR *)
      END (* IF *)
END (* procedure get_data *);


PROCEDURE WRITE_DATA;

(* stampa i risultati *)

VAR
   I : INTEGER;
```

Figura 5.1 — Parte iniziale di un programma di interpolazione di una curva (segue).

```
BEGIN
   WRITELN;
   WRITELN(' I   X    Y');
   FOR I := 1 TO N DO
      WRITELN(I:3, X[I]:8:1, Y[I]:9:2);
   WRITELN
END (* write_data *);

BEGIN (* programma principale *)
   DONE := FALSE;
   REPEAT
      GET_DATA (X, Y, N);
      IF NOT DONE THEN
         BEGIN
            WRITE_DATA;
            (* qui devono essere aggiunte più linee *)
         END
   UNTIL DONE
END.
```

Figura 5.1 — Parte iniziale di un programma di interpolazione di una curva (segue).

Studiamo ora più da vicino la procedura GET-DATA e l'algoritmo che usa per simulare dati sperimentali.

L'algoritmo di "scattering" (sparpagliamento)

La procedura GET-DATA genera una linea retta con un'intercetta (A) di 2 e una tangente (B) di 5, cioè un insieme di dati nelle matrici X e Y che corrispondono alla retta:

$$y = 2 + 5x$$

```
PROGRAM CFIT1A(INPUT,OUTPUT);
(* Programma Pascal che esegue una *)
(* interpolazione lineare coi minimi quadrati *)
(* 22 Ott 80 *)
CONST
   MAX = 20;

TYPE
   INDEX = 1..MAX;
   ARY   = ARRAY[INDEX] OF REAL;

VAR
  X, Y, Y_CALC : ARY;
  N       : INTEGER;
  DONE    : BOOLEAN;
  SEED, A, B : REAL;

FUNCTION RANDOM(DUMMY: INTEGER): REAL;
(* numero random 0 - 1 *)
(* definisce SEED = 4.0 come globale *)
CONST
   PI = 3.14159;

VAR
   X: REAL;
   I : INTEGER;

BEGIN  (* Random *)
   X := SEED + PI;
   X := EXP(5.0 * LN(X));
   SEED := X - TRUNC(X);
   RANDOM := SEED
END (* Random *);
```

Figura 5.2 — Routine alternativa con la procedura RANDOM.

```
PROCEDURE GET_DATA(VAR X, Y : ARY;
                   VAR N : INTEGER);

(* riceve i valori di n e delle matrici x, y *)
(* y è sparpagliato in modo random su una linea retta *)

CONST
  A = 2.0;
  B = 5.0;

VAR
  I, J     : INTEGER;
  FUDGE : REAL;

BEGIN
  WRITE ('Falsificazione?');
  READLN( FUDGE);
  IF FUDGE < 0.0 THEN DONE := TRUE
  ELSE
    BEGIN
      REPEAT
        WRITE ('Quanti punti?');
        READLN( N)
      UNTIL (N > 2) AND (N <= MAX);
      FOR I := 1 TO N DO
        BEGIN
          J := N + 1 - I;
          X[I] := J;
          Y[I] := (A + B*J)
                  *(1.0 + (2.0*RANDOM(0) - 1.0)*FUDGE)
        END  (* ciclo FOR *)
    END  (* IF *)
END; (* procedura get_data *)
```

Figura 5.2 — Routine alternativa con la procedura RANDOM (segue).

```
PROCEDURE WRITE_DATA;
(* stampa i risultati *)
VAR
   I : INTEGER;
BEGIN
   WRITELN;
   WRITELN(' I   X    Y');
   FOR I := 1 TO N DO
      WRITELN(I:3, X[I]:8:1, Y[I]:9:2);
   WRITELN
END (* write_data *);
BEGIN (* programma principale *)
   SEED := 4.0;
   DONE := FALSE;
   REPEAT
      GET_DATA (X, Y, N);
      IF NOT DONE THEN
         BEGIN
            WRITE_DATA;
            (* qui devono essere aggiunte più linee *)
         END
   UNTIL DONE
END.
```

Figura 5.2 — Routine alternativa con la procedura RANDOM (segue).

Si usa una funzione di generazione di numeri casuali per allontanare i punti dalla retta in base al valore della variabile FUDGE. Se FUDGE è zero viene generata una linea perfettamente retta; se FUDGE è 0.2 i punti si allontaneranno al massimo del 20% dalla retta.

Sarebbe più realistico usare la funzione di generazione di numeri casuali Gaussia-

na, piuttosto di quella uniforme, ma il metodo che abbiamo scelto è il più rapido; inoltre questa parte del programma sarà eliminata quando nella routine di interpolazione saranno inseriti dati reali.

L'algoritmo di "scattering" opera nel modo seguente: la funzione RANDOM restituisce un numero reale compreso tra 0 e 1, che viene raddoppiato per avere un intervallo tra 0 e 2. Sottraendo 1 l'intervallo si sposta tra −1 e +1; il risultato viene infine moltiplicato per FUDGE e sommato a 1 per ottenere l'intervallo desiderato.

Notate che X, Y e N vengono passati alla procedura GET-DATA tramite una lista di parametri formali; in questo caso non sarebbe strettamente necessario, perchè la procedura potrebbe ottenere i valori come variabili globali. D'altra parte questo metodo isola effettivamente i dati di ingresso dal programma principale.

Supponiamo ad esempio di voler eseguire una trasformazione dopo aver letto i valori delle matrici X e Y, del tipo:

$$\ln y = A + \frac{B}{x}$$

La lista di parametri "dummy" della procedura GET-DATA dovrebbe contenere i valori trasformati, e l'intestazione dovrebbe essere:

```
PROCEDURE GET-DATA (VAR XR, LOGY : ARY; VAR N: INTEGER);
```

La procedura WRITE-DATA, invece, non ha parametri formali.

Esecuzione del programma principale

Compilate il programma e provatelo; vi chiederà di inserire un valore per la variabile FUDGE. La prima volta rispondete con uno zero. Quando vi chiederà il numero di punti, rispondete 9; il programma stamperà tre colonne di numeri, come in Figura 5.3. Notate che gli elementi della matrice X sono generati in ordine decrescente; riordineremo le matrici X e Y quando aggiungeremo una routine di ordinamento, nel prossimo capitolo.

Alla fine il programma chiede un altro valore di FUDGE; questa volta rispondete 0.2. I valori di x saranno gli stessi dell'esecuzione precedente, mentre quelli di y saranno più grandi o piccoli. Alla fine di questa nuova esecuzione inserite un numero negativo per FUDGE, in modo di uscire dal programma e restituire il controllo al sistema.

```
I        X        Y
1       9.0     47.00
2       8.0     42.00
3       7.0     37.00
4       6.0     32.00
5       5.0     27.00
6       4.0     22.00
7       3.0     17.00
8       2.0     12.00
9       1.0      7.00
```

Figura 5.3 — Prima esecuzione del programma di interpolazione di una curva.

Ora potete aggiungere al programma nuove parti, osservando, però, sempre la copia delle versioni precedenti; così, se avete problemi con le nuove versioni, potete tornare alle precedenti e ripartire da capo.

UNA ROUTINE DI STAMPA DI UN GRAFICO

Aggiungeremo ora una routine per stampare il grafico dei risultati su di un normale terminale di un calcolatore. I grossi calcolatori hanno, in genere, plotter digitali o addirittura teminali video grafici, che consentono una rappresentazione grafica dei dati con un alto grado di precisione, ma con grande dispendio di tempo; inoltre i piccoli calcolatori non hanno in genere questi dispositivi.

In alternativa, si possono ottenere grafici su normali terminali o su una stampante, (usando i simboli (+) e (*). Il grafico risultante è una rappresentazione un po' rozza dei dati reali, ma può essere utile per trovare errori macroscopici nel programma o nei dati di ingresso; inoltre è ottenuto immediatamente, insieme ai risultati dell'elaborazione, mentre in genere è necessario parecchio tempo per avere la stampa di un grafico su un plotter, da parte di un grosso calcolatore.

Nella Figura 5.4 c'è una procedura Pascal per stampare il grafico di una o due variabili dipendenti al variare di una terza variabile indipendente; la routine stampa la variabile indipendente su una retta verticale invece che orizzontale, come nei grafici normali, e la variabile dipendente su una retta orizzontale. Il grafico è dunque ruotato di 90° in senso orario rispetto ad un normale sistema di assi cartesiani. Noi stamperemo il grafico di funzioni semplici, ma si possono visualizzare anche funzioni a molti valori; inoltre si può far variare la variabile indipendente in modo non uniforme.

```
PROCEDURE PLOT(   (* con matrici *)
                  X, (* variabile indipendente *)
                  Y, (* variabile dipendente *) *)
                  YCALC (* curva interpolata *) : ARY;
   (* e *) M : INTEGER (* numero di punti *);

(* disegna y e ycalc come una funzione di x per m punti *)
(* se m è negativo, sono disegnati solo x e y *)
(* 9 Feb 1981 *)

CONST
   BLANK = ' ';
   LINEL = 51;

VAR
   YLABEL: ARRAY[1..6] OF REAL;
   OUT  : ARRAY[1..LINEL] OF CHAR;
   LINES, I, J, JP, L, N: INTEGER;
   ISKIP, YONLY: BOOLEAN;
   XLOW, XHIGH, XNEXT, XLABEL, XSCALE, SIGNXS,
   YMIN, YMAX, CHANGE, YSCALE, YS10: REAL;

FUNCTION PSCALE(P: REAL): INTEGER;

BEGIN
   PSCALE := TRUNC((P - YMIN)/YSCALE + 1)
END (* pscale *);

  PROCEDURE OUTLIN(XNAME: REAL);
  (* stampa una linea *)

  VAR
     I, MAX: INTEGER;
```

Figura 5.4 — Procedura di stampa di un grafico.

```
BEGIN
   WRITE(XNAME: 8: 2, BLANK) (* linea di etichetta *);
   MAX := LINEL + 1;
   REPEAT    (* salta i blank alla fine della linea *)
      MAX := MAX - 1
   UNTIL (OUT[MAX] <> BLANK) OR (MAX = 1);
   FOR I := 1 TO MAX DO
      WRITE(OUT[I]);
   WRITELN;
   FOR I := 1 TO MAX DO
      OUT[I] := BLANK     (* prossima linea bianca *)
END (* outlin *);

PROCEDURE SETUP(INDEX: INTEGER);
(* pone il più e l'asterisco per la stampa *)

CONST
   STAR = '*';
   PLUS = '+';

VAR
   I: INTEGER;

BEGIN
   I := PSCALE(Y[INDEX]);
   OUT[I] := PLUS;

  IF NOT YONLY THEN
     BEGIN     (* aggiunte anche ycalc *)
        I := PSCALE(YCALC[INDEX]);
        OUT[I] := STAR
     END
END   (* setup *);
```

Figura 5.4 — Procedura di stampa di un grafico (segue).

```
BEGIN      (* corpo del tracciato *)
  IF M > 0 THEN ;(* traccia y e ycalc su x *)
    BEGIN
      N := M;
      YONLY := FALSE
    END
  ELSE     (* traccia solo y su x *)
    BEGIN
      N := - M;
      YONLY := TRUE
    END;
  (* spazia a linee alternate *)
  LINES := 2 * (N - 1) + 1;
  WRITELN;
  XLOW := X[1];
  XHIGH := X[N];
  YMAX := Y[1];
  YMIN := YMAX;
  XSCALE := (XHIGH - XLOW)/(LINES - 1);
  SIGNXS := 1.0;
  IF XSCALE < 0.0 THEN SIGNXS := -1.0;
  FOR I := 1 TO N DO
    BEGIN
      IF Y[I] < YMIN THEN  YMIN := Y[I];
      IF Y[I] > YMAX THEN  YMAX := Y[I];

      IF NOT YONLY THEN
        BEGIN
          IF YCALC[I] < YMIN THEN  YMIN := YCALC[I];
          IF YCALC[I] > YMAX THEN  YMAX := YCALC[I]
        END    (* IF yonly *)
```

Figura 5.4 — Procedura di stampa di un grafico (segue).

```
   END;
YSCALE := (YMAX − YMIN)/(LINEL − 1);
YS10 := YSCALE * 10;
YLABEL[1] := YMIN    (* asse y *);
FOR I := 1 TO 4 DO
   YLABEL[I + 1] := YLABEL[I] + YS10;
YLABEL[6] := YMAX;
FOR I := 1 TO LINEL DO
   OUT[I] := BLANK    (* linea bianca *);
SETUP(1);
L := 1;
XLABEL := XLOW;
ISKIP := FALSE;

FOR I := 2 TO LINES DO    (* sistema una linea *)
   BEGIN
      XNEXT := XLOW + XSCALE * (I − 1);
      IF ISKIP THEN WRITELN('  −')
      ELSE
         BEGIN
            L := L + 1;
            WHILE
               (X[L] − (XNEXT − 0.5 * XSCALE))*SIGNXS
                  <= 0.0  DO
               BEGIN
                  SETUP(L)  (* sistema la linea di stampa *);
                  L := L + 1
               END (* WHILE *);

               OUTLIN(XLABEL)    (* stampa una linea *);
               FOR J := 1 TO LINEL DO
                  OUT[J] := BLANK    (* linea bianca *)
         END    (* IF iskip *);
```

Figura 5.4 — Procedura di stampa di un grafico (segue).

```
        IF (X[L] — (XNEXT + 0.5 * XSCALE))*SIGNXS > 0.0
          THEN ISKIP := TRUE
        ELSE
          BEGIN
            ISKIP := FALSE;
            XLABEL := XNEXT;
            SETUP(L)   (* sistema la linea di stampa *)
          END
      END    (* ciclo FOR *);
    OUTLIN(XHIGH) (* ultima linea *);
    WRITE(' ');
    FOR I := 1 TO 6 DO
      WRITE('  ^  ');
    WRITELN;
    WRITE(' ');
    FOR I := 1 TO 6 DO
      WRITE(YLABEL[I]: 9: 1, BLANK);
    WRITELN;
    WRITELN
  END  (* plot *);
```

Figura 5.4 — Procedura di stampa di un grafico (segue).

Esecuzione della routine di stampa del grafico

Fate una copia dell'ultima versione del programma sorgente, e dategli il nome CFIT2.PAS. Aggiungete la procedura PLOT subito dopo la WRITE-DATA e prima dell'istruzione BEGIN del programma principale, e inserite nel programma principale una chiamata alla nuova procedura:

```
PLOT(X, Y, Y, -N);
```

immediatamente dopo la chiamata alla procedura WRITE-DATA e prima dell'istruzione UNTIL. Assicuratevi che ci sia il punto e virgola tra le chiamate alla WRITE-DATA e alla PLOT; questo non è invece necessario alla fine della chiamata alla PLOT, ma semplifica le aggiunte successive al programma. Notate che il secondo e il terzo argomento della chiamata alla PLOT sono identici, e che l'ultimo è negativo, per segnalare che deve essere stampato il grafico di una sola curva.

Compilate il programma e provatelo. Se date un valore di 0.2 per FUDGE l'uscita sarà come in Figura 5.5; se invece date il valore 0, avrete una linea quasi retta.

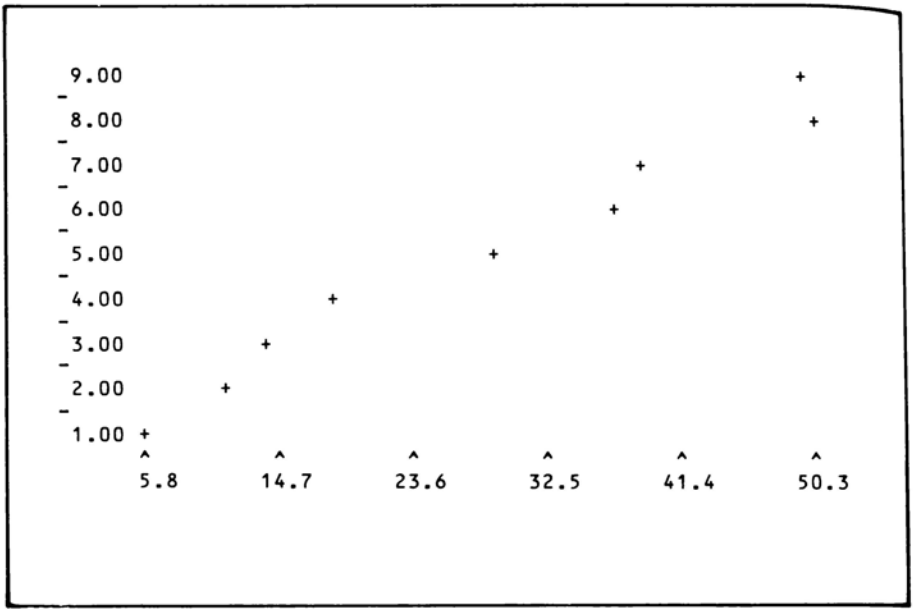


Figura 5.5 — Grafico di y in funzione di x.

La routine PLOT sarà inserita in tutte le prossime versioni del programma; visto che il programma è particolarmente lungo, può convenire registrarla su di un file separato, richiamandola con una direttiva INCLUDE o con un'istruzione EXTERN, entrambe descritte nel Capitolo 1.

Per provare tutti gli aspetti di questa routine, possiamo cercare di stampare il grafico di due curve contemporaneamente, aggiungendo la breve procedura LINFIT nel nostro programma. In seguito questa sarà sostituita da un'altra procedura, più significativa, quando avremo studiato l'algoritmo di interpolazione lineare di una curva con i minimi quadrati.

SIMULAZIONE DI UNA ROUTINE DI INTERPOLAZIONE DI UNA CURVA

Aggiungiamo ora al programma una routine simulata di interpolazione di una curva, che genera semplicemente un vettore Y_CALC giacente sulla nostra retta con tangente 5 e intercetta 2. In una prossima versione aggiungeremo una vera e propria routine di interpolazione.

Aggiungete la procedura LINFIT della Figura 5.6 al programma sorgente, inserendola immediatamente dopo la WRITE_DATA. Dovrete fare alcune modifiche al programma principale e alla procedura WRITE_DATA, come mostrato nelle Figure 5.8 e 5.7; notate che Y_CALC è il terzo parametro nella chiamata alla PLOT, e che l'ultimo parametro è positivo.

```
PROCEDURE LINFIT(X, Y : ARY;
          VAR Y_CALC : ARY;
             VAR A, B  : REAL;
                     N : INTEGER);

(* genera una linea retta per x-y *)

VAR
  I : INTEGER;

BEGIN (* linfit *)
  A := 2.0;
  B := 5.0;
  FOR I := 1 TO N DO
        Y_CALC[I] := A + B * X[I]
END (* linfit *);
```

Figura 5.6 — Procedura LINFIT di simulazione di un'interpolazione lineare.

```
PROCEDURE WRITE_DATA;

(* stampa i risultati *)

VAR
   I : INTEGER;

BEGIN
   WRITELN;
   WRITELN(' I   X    Y    Y CALC');
   FOR I := 1 TO N DO
      WRITELN(I:3, X[I]:8:1, Y[I]:9:2, Y_CALC[I]:9:2);
   WRITELN
END (* write_data *);
```

Figura 5.7 — La procedura WRITE_DATA modificata.

```
BEGIN (* programma principale *)
   DONE := FALSE;
   REPEAT
      GET_DATA (X, Y, N);
      IF NOT DONE THEN
         BEGIN
            LINFIT(X, Y, Y_CALC, A, B, N);
            WRITE_DATA;
            PLOT (X, Y, Y_CALC, N)
         END
   UNTIL DONE
END.
```

Figura 5.8 — Programma principale.

Esecuzione della routine di stampa del grafico per due curve

Compilate la nuova versione e provatela. Quando il programma vi chiederà un valore per FUDGE rispondete 0.2; otterrete quattro colonne di dati, fra le quali il vettore Y_CALC. Viene poi stampato un grafico di due diverse curve: gli asterischi (*) rappresentano i valori di Y_CALC, e formano una retta quasi perfetta. I (+) rappresentano i valori di y, che sono distribuiti intorno a Y_CALC, come si vede dalla Figura 5.9. Se, per un certo valore di x, i simboli coincidono, viene stampato solo l'asterisco.

Quando il programma vi chiede un altro valore di FUDGE, rispondete zero; y e Y_CALC avranno lo stesso valore, le due linee si sovrappongono e ne viene stampata solo una. Alla fine date un valore negativo per FUDGE per uscire dal programma.

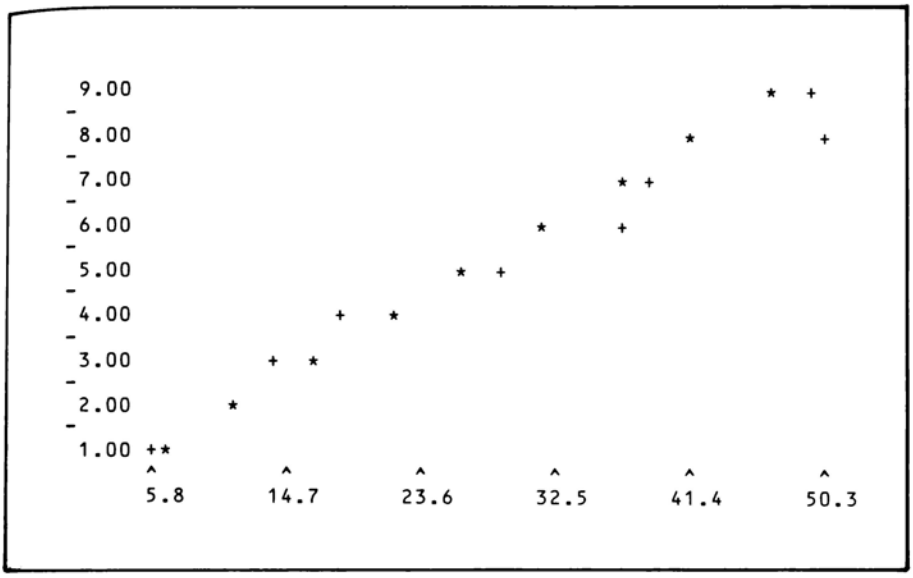


Figura 5.9 — Grafico di y e Y_CALC in funzione di x.

Ora che abbiamo scritto e provato sia il programma principale che la procedura per stampare i grafici, possiamo passare all'argomento centrale del capitolo.

L'ALGORITMO DI INTERPOLAZIONE DI UNA CURVA

Anche se dalla Figura 5.9 potrebbe sembrare che abbiamo completato il nostro programma, in realtà non è così; per ora abbiamo solo generato i valori di Y_CALC

che corrispondono alla nostra linea originale. Ora vogliamo derivare l'algoritmo per una procedura di interpolazione lineare coi minimi quadrati. Cominceremo con l'introdurre un nuovo vettore r che contiene i residui.

Ad ogni punto sperimentale, di coordinate x-y, corrisponde un elemento di r che rappresenta la differenza tra il valore calcolato di Y_CALC (che indicheremo con $\hat{y}$) e il valore originale di y. Possiamo esprimere questa relazione, in termini matematici, come:

$$r_i = \hat{y}_i - y_i \tag{1}$$

Può capitare che un punto cada sulla curva calcolata, ma, in generale, circa metà dei punti saranno da una parte rispetto alla curva calcolata, e, in corrispondenza di questi, il vettore r conterrà valori positivi. Gli altri si troveranno dall'altra parte della curva, e corrisponderanno a valori negativi di r. La somma dei residui dovrebbe essere prossima a zero.

Il criterio di interpolazione coi minimi quadrati è che la somma dei quadrati dei residui deve essere minimizzata. Il quadrato di ogni residuo sarà positivo, e così anche la somma dei quadrati (SRS). Questo criterio può essere espresso come:

$$SRS = \sum_{i=1}^{n} r_i^2 = \text{minimum} \tag{2}$$

dove n è il numero dei punti x-y (e la lunghezza dei vettori x, y e $\hat{y}$).

Combinando l'equazione 1 con quella di interpolazione:

$$\hat{y}_i = A + Bx_i \tag{3}$$

abbiamo:

$$r_i = A + Bx_i - y_i \tag{4}$$

e:

$$SRS = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (A + Bx_i - y_i)^2 \tag{5}$$

Il problema si riduce nel trovare i valori di A e B che minimizzano la sommatoria dell'equazione 5; si può fare in diversi modi. Noi faremo la derivata dell'equazione 5 rispetto alle due variabili A e B, e porremo il risultato a zero.

$$\frac{\delta \Sigma r_i^2}{\delta A} = 0 \quad \text{and} \quad \frac{\delta \Sigma r_i^2}{\delta B} = 0 \tag{6}$$

Sostituendo l'equazione 5 nella 6 otteniamo:

$$\frac{\delta\Sigma\,(A + Bx_i - y_i)^2}{\delta B} = 0 \tag{7}$$

e

$$\frac{\delta\Sigma\,(A + Bx_i - y_i)^2}{\delta A} = 0 \tag{8}$$

che è equivalente a:

$$\frac{2\Sigma\,(A + Bx_i - y_i)\,\delta\Sigma\,(A + Bx_i - y_i)}{\delta A} = 0 \tag{9}$$

e

$$\frac{2\Sigma\,(A + Bx_i - y_i)\,\delta\Sigma\,(A + Bx_i - y_i)}{\delta B} = 0 \tag{10}$$

Poichè B, x e y non dipendono da A, e la derivata di A rispetto a se stessa è 1, l'equazione 9 diventa:

$$\Sigma A + \Sigma Bx_i = \Sigma y_i \tag{11}$$

Analogamente A, x e y non dipendono da B, e l'equazione 10 diventa:

$$\Sigma Ax_i + \Sigma Bx_i^2 = \Sigma x_i y_i \tag{12}$$

A e B sono costanti, e possono essere messi a fattor comune nelle sommatorie. Le equazioni 7 e 8 diventano allora:

$$An + B\Sigma x_i = \Sigma y_i \tag{13}$$

e

$$A\Sigma x_i + B\Sigma x_i^2 = \Sigma x_i y_i \tag{14}$$

Abbiamo riportato il problema di trovare una retta che passa per un insieme di punti x-y dati alla risoluzione di un sistema di due equazioni (13 e 14) lineari nelle due incognite A e B (x, y e n sono i dati originali). La soluzione del sistema di trova con la regola di Cramer:

$$A = \frac{\begin{vmatrix} \Sigma y_i & \Sigma x_i \\ \Sigma x_i y_i & \Sigma y_i^2 \end{vmatrix}}{\begin{vmatrix} n & \Sigma x_i \\ \Sigma x_i & \Sigma x_i^2 \end{vmatrix}} \tag{15}$$

e

$$B = \frac{\begin{vmatrix} n & \Sigma y_i \\ \Sigma x_i & \Sigma x_i y_i \end{vmatrix}}{\begin{vmatrix} n & \Sigma x_i \\ \Sigma x_i & \Sigma x_i^2 \end{vmatrix}} \tag{16}$$

che corrispondono a:

$$A = \frac{\Sigma x_i^2 \Sigma y_i - \Sigma x_i \Sigma x_i y_i}{n \Sigma x_i^2 - \Sigma x_i \Sigma x_i} \tag{17}$$

e

$$B = \frac{n \Sigma x_i y_i - \Sigma x_i \Sigma y_i}{n \Sigma x_i^2 - \Sigma x_i \Sigma y_i} \tag{18}$$

Il calcolo di A e B è immediato con il calcolatore; la sommatoria di x si ottiene sommando i valori della matrice X; la sommatoria di x^2 si ottiene elevando al quadrato ogni valore della matrice X e sommando i risultati.

Le equazioni 17 e 18 si trasformano in:

$$A = \frac{(\Sigma x_i^2 \Sigma y_i - \Sigma x_i \Sigma x_i y_i)/n}{\Sigma x_i^2 - \Sigma x_i \Sigma x_i / n} \tag{19}$$

e

$$B = \frac{\Sigma x_i y_i - \Sigma x_i \Sigma y_i / n}{\Sigma x_i^2 - \Sigma x_i \Sigma x_i / n} \tag{20}$$

dividendo il numeratore e il denominatore per n.

I denominatori delle equazioni 19 e 20 compaiono nella formula della deviazione standard del Capitolo 2.

Abbiamo richiamato i concetti matematici del metodo di interpolazione coi minimi quadrati e abbiamo ricavato le formule per trovare la tangente (B) e l'intercetta (A) dell'interpolazione lineare di una curva; ora siamo in grado di affrontare la procedura di interpolazione LINFIT.

La procedura di interpolazione di una curva

In Figura 5.10 è presentata una procedura per interpolare una retta; sostituitela alla versione originale di LINFIT.

```
PROCEDURE LINFIT(X, Y : ARY;
                 VAR Y_CALC : ARY;
                 VAR A, B       : REAL;
                         N      : INTEGER);
(* interpola una linea retta (y-calc) attraverso
   n coppie di punti x e y *)
VAR
  I : INTEGER;
  SUM_X, SUM_Y, SUM_XY, SUM_X2,
  SUM_Y2, XI, YI, SXY ,SXX, SYY : REAL;
BEGIN (* linfit *)
  SUM_X := 0.0;
  SUM_Y := 0.0;
  SUM_XY := 0.0;
  SUM_X2 := 0.0;
  SUM_Y2 := 0.0;
  FOR I := 1 TO N DO
    BEGIN
      XI := X[I];
      YI := Y[I];
      SUM_X := SUM_X + XI;
      SUM_Y := SUM_Y + YI;
      SUM_XY := SUM_XY + XI * YI;
      SUM_X2 := SUM_X2 + XI * XI;
      SUM_Y2 := SUM_Y2 + YI * YI
    END;
  SXX := SUM_X2 - SUM_X * SUM_X/ N;
  SXY := SUM_XY - SUM_X * SUM_Y/ N;
  SYY := SUM_Y2 - SUM_Y * SUM_Y/ N;
  B := SXY/SXX;
  A := ((SUM_X2 * SUM_Y - SUM_X * SUM_XY)/ N)/SXX;
  FOR I := 1 TO N DO
    Y_CALC[I] := A + B * X[I]
END (* linfit *);
```

Figura 5.10 — Procedura LINFIT per interpolazione con i minimi quadrati.

All'inizio della procedura, alle variabili SUM_X, SUM_Y etc. viene assegnato il valore 0, e poi viene usato un ciclo FOR per calcolare le sommatorie. Notate che all'inizio del ciclo viene fatto un cambiamento di variabile:

```
XI : = X[I];
YI : = Y[I];
```

In generale richiede più tempo usare un elemento di una matrice piuttosto che uno scalare; perciò, quando uno stesso elemento di una matrice viene usato molte volte in un ciclo, conviene definire una variabile scalare che lo contiene, e usare questa al suo posto. Per altro, alcuni moderni compilatori eseguono automaticamente questa funzione, mediante un ottimizzatore; in questo caso ovviamente non è necessario eseguire la trasformazione precedente.

Le matrici X, Y e Y_CALC e le variabili scalari A, B e N sono parametri formali come prima. In questo modo possiamo richiamare LINFIT in un certo punto del programma:

```
LINFIT(X, Y, Y_CALC, A, B, N);
```

per interpolare X e Y, e richiamarla in un punto successivo per interpolare X1 e Y1 con un diverso numero di punti:

```
LINFIT(X1, Y1, Y1C, A1, B1, M);
```

Con questa seconda chiamata viene interpolata una retta con tangente B1 e intercetta A1 con i punti di X1 e Y1.

Esecuzione del programma di interpolazione di una curva

Compilate la nuova versione e provatela. Cominciate col dare a FUDGE il valore 0.2. La retta di asterischi dovrebbe passare attraverso la curva dei "+". Confrontate la figura 5.11, che mostra l'interpolazione effettiva, con la 5.9, che mostra quella simulata. Poi date a FUDGE il valore zero; verrà stampata un'unica linea di asterischi che rappresenta una retta di intercetta 2 e tangente 5.

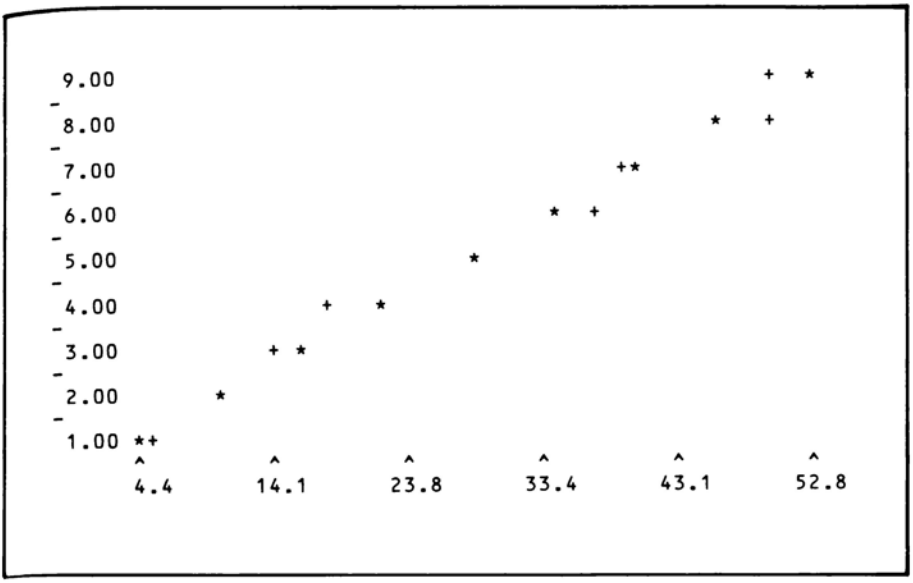


Figura 5.11 — Interpolazione con i minimi quadrati.

IL COEFFICIENTE DI CORRELAZIONE

Ora abbiamo una procedura per calcolare i punti della retta desiderata, ma non abbiamo ancora finito. Possiamo ottenere l'equazione di una curva che interpola i nostri dati sperimentali, e usarla per prevedere il valore di y dato un certo valore di x. In certi casi, però, la nostra soluzione anche se matematicamente corretta, è priva di utilità.

Consideriamo ad esempio i dati dei due grafici della Figura 5.12; in entrambi i casi la procedura LINFIT troverà l'equazione di una retta, ma la curva risultante non ci dà nessuna informazione, in quanto l'andamento di x non ci dice niente su quello di y. Non c'è correlazione tra x e y.

I dati di Figura 5.13 rappresentano un altro caso in cui la conoscenza di x non è di nessun aiuto per prevedere l'andamento di y; anche qui non c'è nessuna correlazione tra x e y.

Abbiamo bisogno di una misura quantitativa della correlazione tra x e y, per sapere con quanta precisione possiamo prevedere l'andamento di y se conosciamo quello di x; questa misura è il coefficiente di correlazione.

Abbiamo visto nel Capitolo 2 che possiamo caratterizzare un insieme di dati con la media e la deviazione standard. Possiamo anche calcolare la deviazione standard

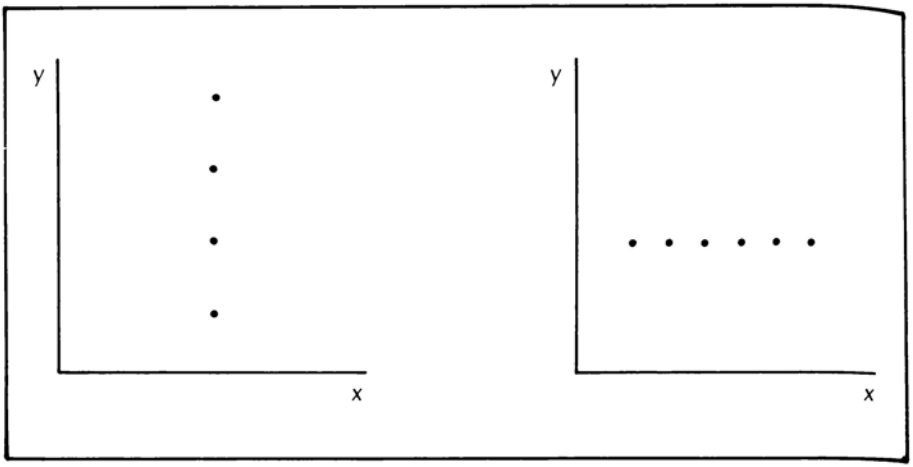


Figura 5.12 — Quando non c'è correlazione tra x e y.

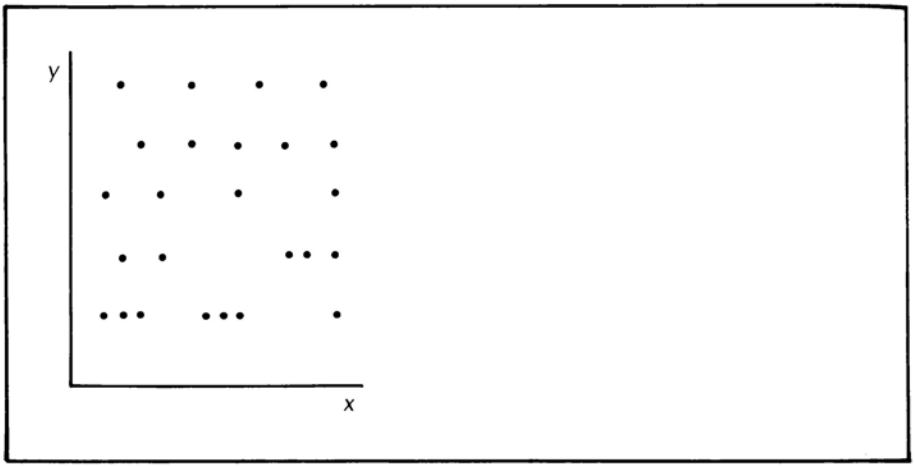


Figura 5.13 — Dati molto dispersi.

di y sulla curva interpolata, che si chiama errore standard di stima (SEE). Il coefficiente di correlazione confronta la deviazione standard di y (intorno al valor medio) con la deviazione standard intorno alla curva interpolata (SEE).

Il coefficiente di correlazione è zero se non c'è correlazione, come nel caso dei dati delle Figure 5.12 e 5.13. D'altra parte il coefficiente di correlazione si avvicina tanto più a 1 quando più i dati si avvicinano ad una linea retta. Il coefficiente di correlazione dei dati della Figura 5.11 è 0.99.

Aggiungeremo ora, nel nostro programma, il calcolo del coefficiente di correlazione.

Programma Pascal per calcolare il Coefficiente di Correlazione

Con l'aggiunta di poche istruzioni, il nostro programma può calcolare il coefficiente di correlazione e gli errori standard dei coefficienti (l'intercetta e la tangente). Gli errori standard sono le deviazioni standard dei coefficienti, e si usano per determinare gli intervalli di confidenza per ogni coefficiente.

Aggiungete l'espressione:

```
CORREL_COEF, SIGMA_A, SIGMA_B, SEE : REAL;
```

nella sezione VAR che si trova all'inizio del programma; saranno variabili globali. Le quattro linee seguenti sono inserite nella procedura LINFIT subito dopo la definizione dell'intercetta A:

```
CORREL_COEF : =
  SXY/SQRT(SXX*SYY);
SEE : =
  SQRT ( (SUM_Y2_A*SUM_Y_B*SUM_XY) / (N_2) );
SIGMA_B : = SEE/SQRT(SXX);
SIGMA_A : = SIGMA_B*SQRT(SUM_X2/N);
```

Infine le istruzioni di uscita della procedura WRITE_DATA sono sostituite dalle seguenti:

```
WRITELN('l'intercetta è', A: 8: 3,
        'sigma è', SIGMA_A:8:3);
WRITELN('la pendenza è',B:8:2,
        'sigma è',SIGMA_B:8:3);
WRITELN
WRITELN('il coefficiente di correlazione è',
        CORREL_COEFF:7:4)
```

Con il coefficiente di correlazione abbiamo fornito un'ulteriore possibilità al nostro programma, quella di quantificare l'utilità della curva interpolata. Vediamo ora come si presenta il programma completo.

PROGRAMMA PASCAL: INTERPOLAZIONE DI UNA CURVA CON I MINIMI QUADRATI PER UN INSIEME DI DATI SIMULATI

Compilate la nuova versione e provatela. Date a FUDGE il valore 0, otterrete intercetta 2 e tangente 5, come prima; inoltre, i valori di sigma per A e B saranno 0 e il coefficiente di correlazione 1. Il grafico rappresenterà una retta di asterischi.

Date a FUDGE il valore 0.2; avrete dei valori di sigma maggiori di zero per l'intercetta e la tangente, e un coefficiente di correlazione minore di 1. In Figura 5.14 c'è il programma completo.

```
PROGRAM CFIT4(INPUT,OUTPUT);
(* Programma Pascal che esegue *)
(* l'interpolazione lineare ai minimi quadrati *)
(* 8 Feb 81 *)
CONST
   MAX = 20;
TYPE
   INDEX = 1..MAX;
   ARY   = ARRAY[INDEX] OF REAL;
VAR
   X, Y, Y_CALC : ARY;
   N        : INTEGER;
   DONE  : BOOLEAN;
   A, B, CORREL_COEF,
   SIGMA_A, SIGMA_B, SEE : REAL;

PROCEDURE GET_DATA(VAR X, Y : ARY;
                   VAR N : INTEGER);
(* riceve i valori di n e delle matrici x, y *)
(* y è sparpagliato in modo random su una linea retta *)
CONST
   A = 2.0;
   B = 5.0;

VAR
   I, J : INTEGER;
   FUDGE : REAL;

BEGIN
   WRITE ('Falsificazione?');
   READLN( FUDGE);
   IF FUDGE < 0.0 THEN DONE := TRUE
   ELSE
```

Figura 5.14 — Il programma completo di interpolazione di una curva.

```
      BEGIN
        REPEAT
          WRITE (´Quanti punti?´);
          READLN( N)
        UNTIL (N > 2) AND (N <= MAX);
        FOR I := 1 TO N DO
          BEGIN
            J := N + 1 - I;
            X[I] := J;
            Y[I] := (A + B*J)
              *(1.0 + (2.0*RANDOM(0) - 1.0)*FUDGE)
          END (* ciclo FOR *)
      END (* ELSE *)
END (* procedura get_data *);
PROCEDURE WRITE_DATA;
(* stampa i risultati *)
VAR
  I : INTEGER;
BEGIN
  WRITELN;
  WRITELN(' I    X    Y   Y CALC');
  FOR I := 1 TO N DO
    WRITELN(I:3, X[I]:8:1,
            Y[I]:9:2, Y_CALC[I]:9:2);
  WRITELN;
  WRITELN (´l'intercetta è´, A :8:3,
          ´, sigma è´, SIGMA_A :8:3);
  WRITELN (´ la pendenza è´, B :8:2,
          ´, sigma è´, SIGMA_B :8:3);
  WRITELN;
  WRITELN (´Il coefficiente di correlazione è´,
           CORREL_COEFF :7:4)
END (* write_data *);
```

Figura 5.14 — Il programma completo di interpolazione di una curva (segue).

```
PROCEDURE LINFIT(X, Y : ARY;
          VAR Y_CALC: ARY;
          VAR A, B    : REAL;
                    N : INTEGER);
(* interpola una linea retta (y-calc) attraverso
 n coppie di punti x e y *)

VAR
   I : INTEGER;
   SUM_X, SUM_Y, SUM_XY, SUM_X2,
   SUM_Y2, XI, YI, SXY ,SXX, SYY : REAL;


BEGIN (* linfit *)
   SUM_X := 0.0;
   SUM_Y := 0.0;
   SUM_XY := 0.0;
   SUM_X2 := 0.0;
   SUM_Y2 := 0.0;
   FOR I := 1 TO N DO
      BEGIN
         XI := X[I]; YI := Y[I];
         SUM_X := SUM_X + XI;
         SUM_Y := SUM_Y + YI;
         SUM_XY := SUM_XY + XI * YI;
         SUM_X2 := SUM_X2 + XI * XI;
         SUM_Y2 := SUM_Y2 + YI * YI
      END;
   SXX := SUM_X2 - SUM_X * SUM_X / N;
   SXY := SUM_XY - SUM_X * SUM_Y / N;
   SYY := SUM_Y2 - SUM_Y * SUM_Y / N;
   B := SXY/SXX;
   A := ((SUM_X2 * SUM_Y - SUM_X
                  * SUM_XY)/ N)/SXX;
```

Figura 5.14 — Il programma completo di interpolazione di una curva (segue).

```
  CORREL_COEF :=
          SXY/ SQRT( SXX * SYY);
  SEE :=
       SQRT( (SUM_Y2 - A*SUM_Y - B*SUM_XY)
                                  /(N - 2));
  SIGMA_B := SEE/SQRT(SXX);
  SIGMA_A := SIGMA_B * SQRT( SUM_X2/N);
  FOR I := 1 TO N DO
     Y_CALC[I] := A + B * X[I]
END (* linfit *);


(* PROCEDURE plot
          (x, y, y_calc : ary;
                     n : integer);

extern; *)
(*$F PLOT.PAS *)


BEGIN (* programma principale *)
  DONE := FALSE;
  REPEAT
     GET_DATA (X, Y, N);
     IF NOT DONE THEN
        BEGIN
           LINFIT(X, Y, Y_CALC, A, B, N);
           WRITE_DATA;
           PLOT (X, Y, Y_CALC, N)
        END
  UNTIL DONE
END.
```

Figura 5.14 — Il programma completo di interpolazione di una curva (segue).

SOMMARIO

La tecnica di sviluppo modulare usata per questo programma ci ha consentito di provare le singole procedure man mano che venivano scritte. Abbiamo cominciato dal programma principale, e abbiamo aggiunto di volta in volta le procedure per simulare dati, stampare il grafico dei risultati, calcolare la curva interpolata e il coefficiente di correlazione. La procedura di interpolazione LINFIT sostituisce una precedente procedura che era stata scritta solo per provare la routine di stampa del grafico.

Abbiamo ora un programma che esegue l'interpolazione lineare di una curva; purtroppo non serve a un granchè. Infatti può interpolare soltanto dati prodotti da una funzione di generazione di numeri casuali, mentre noi vogliamo modificare la procedura GET_DATA in modo che possa acquisire dati reali da tastiera o da un file su disco. Affronteremo questo problema dopo aver aggiunto una routine di ordinamento, nel prossimo capitolo.

CAPITOLO 6

ORDINAMENTO

INTRODUZIONE

In questo capitolo svilupperemo diversi algoritmi di ordinamento: due ordinamenti a bolle, un ordinamento di Shell, un ordinamento veloce ricorsivo e uno non ricorsivo. Alla fine inseriremo una di queste routine nel programma di interpolazione di una curva del Capitolo 5.

Cominciamo col discutere il motivo per cui è necessario inserire una routine di ordinamento nel nostro programma.

TRATTAMENTO DI DATI SPERIMENTALI

Il programma di interpolazione di una curva che abbiamo scritto nel capitolo precedente acquisiva i dati da una funzione di generazione di numeri casuali; un programma più aderente alla realtà dovrebbe acquisirli da una tastiera o da un file su disco; in alternativa i dati potrebbero far parte del programma stesso.

In particolare potremmo voler interpolare una curva da un insieme di dati sperimentali non in ordine numerico. Supponiamo di voler studiare la dilatazione termica di un materiale, partendo dalla misurazione di diverse temperature e delle corrispondenti lunghezze; portiamo il materiale ad una certa temperatura e ne misuriamo la lunghezza; aumentiamo la temperatura e misuriamo la temperatura del campione e la sua lunghezza. Evidentemente non è opportuno continuare con questo sistema, poichè la lunghezza potrebbe essere misurata prima che la temperatura del pezzo campione sia uniforme; le coppie di valori misurati conterranno tutte un certo errore.

Una migliore tecnica sperimentale sarebbe quella di arrivare alla temperatura

desiderata una volta da valori più alti e un'altra da valori più bassi; le rilevazioni darebbero risultati del tipo:

Temperatura	Lunghezza
100	8.0
300	19.0
200	13.5
500	30.0
400	24.5

Col programma di interpolazione di una curva dell'ultimo capitolo potremmo trovare immediatamente un'interpolazione coi minimi quadrati di questi dati, inserendoli direttamente nella procedura GET_DATA:

```
PROCEDURE GET_DATA...

BEGIN
  N : = 5
  X[1]: = 100; Y[1]: = 8.0;
  X[2]: = 300; Y[2]: = 19.0
  X[3]: = 200; Y[3]: = 13.5
  X[4]: = 500; Y[4]: = 30.0
  X[5]: = 400; Y[5]: = 24.5
END;
```

Se ora eseguiamo il programma, però, ci accorgeremo che la procedura PLOT ci darà una rappresentazione inesatta dei dati, in quanto la matrice della variabile indipendente X deve contenere valori in ordine crescente o decrescente. In sostanza i dati devono essere ordinati. Nel capitolo precedente la PLOT operava correttamente perchè la matrice X era generata in ordine decrescente. Notate invece che non ha importanza l'ordine dei valori della matrice della variabile dipendente Y.

Inizieremo lo studio delle routine di ordinamento da quella che è più semplice da programmare, l'ordinamento a bolla.

ORDINAMENTO A BOLLA

Ordinare un insieme di valori significa metterli in ordine crescente o decrescente; i dati possono essere elementi di un array di numeri reali, oppure un gruppo di caratteri alfanumerici chiamati records. Esistono diversi algoritmi di ordinamento, alcuni molto veloci, altri molto lenti. Alcuni sono molto veloci se i dati sono già pressochè

ordinati, altri invece sono molto lenti in queste condizioni. Alcuni richiedono spazio di lavoro oltre a quello occupato dai dati stessi, altri no.

Il primo algoritmo di ordinamento che studieremo si chiama ordinamento a bolla; è il più facile da capire e da programmare, ma purtroppo è anche il più lento. Per insiemi con una dozzina di elementi la differenza di velocità non è comunque molto importante.

Nell'ordinamento a bolla, ogni elemento viene confrontato con gli altri non ordinati; quando si trova una coppia non ordinata, i due elementi sono scambiati. Ci sono due cicli, uno interno all'altro; quello più esterno va da 1 a N-1, dove N è la lunghezza dell'array; quello più interno va da M+1 (dove M è l'estremo inferiore del ciclo esterno) a N. In questo modo l'elemento più piccolo "galleggia" verso la cima dell'array durante il processo di ordinamento, da cui il nome di ordinamento a bolla.

Inseriremo questa prima routine di ordinamento in un programma studiato per provare l'efficienza di diverse routine di ordinamento sotto diverse condizioni; le altre routine saranno scritte sotto forma di procedure, anche se ognuna potrebbe essere incorporata nel programma TSTSORT.

PROGRAMMA PASCAL: ORDINAMENTO A BOLLA E PROGRAMMA TSTSORT

Il programma della Figura 6.1 contiene una routine "pilota" oltre alla procedura di ordinamento. Quando viene eseguito, chiede all'utente il numero di elementi che devono essere ordinati, e poi chiama una funzione di generazione di numeri casuali; questa genera il numero di dati desiderato, compresi tra 0 e 100, e li stampa sulla console, 10 per riga. L'istruzione CHR(7) fa suonare il campanello della console all'inizio e alla fine del processo di ordinamento, in modo da consentire un confronto tra le diverse routine. Se non avete un terminale ASCII dovete cambiare o eliminare questa espressione.

Alla fine viene stampato sulla console l'array ordinato, insieme con la parola RANDOM; il campanello della console suona una terza volta e viene nuovamente chiamata la procedura di ordinamento. Questa volta però l'ordinamento viene fatto su di un insieme di dati già ordinato. Alla fine di questa seconda fase di ordinamento la campana della console suona una quarta volta e viene stampato nuovamente l'array ordinato, seguito dalla parola 'ordinato'.

Nella terza parte del programma viene generato un array in ordine inverso, e viene chiamata la routine di ordinamento per la terza volta; alla fine viene stampato l'array ordinato seguito dalla parola 'inversi'.

Scrivete il programma della Figura 6.1 ed eseguitelo; cercate un insieme di dati abbastanza numeroso da richiedere alcuni minuti per l'ordinamento, e prendete nota del tempo necessario nei tre casi; in seguito confronterete questa routine di ordinamento con quelle che studieremo più avanti nel capitolo. Se non avete un termi-

nale video, potete eliminare le chiamate alla procedura PRINT in modo che gli array non vengano stampati.

I programmi di questo capitolo usano la funzione RANDOM per generare numeri casuali; se non c'è nel vostro Pascal, inserite nel programma la funzione RANDOM della Figura 5.2 del Capitolo 5.

```
PROGRAM TSTSORT(INPUT, OUTPUT);
(* prova di velocità di una routine di ordinamento *)

CONST
  MAX = 1000;
TYPE
  ARY = ARRAY[1..300] OF REAL;
VAR
  X: ARY;
  N, I: INTEGER;


PROCEDURE PRINT;


VAR
  I: INTEGER;


BEGIN
  WRITELN;
  FOR I := 1 TO N DO
    BEGIN
      WRITE(X[I] :7:2);
      IF (I MOD 10) = 0 THEN WRITELN
    END
END;
```

Figura 6.1 — Una procedura di ordinamento a bolla con programma principale.

```
PROCEDURE (* bubble *) SORT(VAR A: ARY; N: INTEGER);

VAR
   I,J: INTEGER;
   HOLD: REAL;

BEGIN (* procedure sort *)
   FOR I := 1 TO N - 1 DO
      FOR J := I + 1 TO N DO
         BEGIN
            IF A[I] > A[J] THEN
               BEGIN
                  HOLD := A[I];
                  A[I] := A[J];
                  A[J] := HOLD
               END
         END (* FOR *)
END (* procedure sort *);

BEGIN (* programma principale *)
   REPEAT
      REPEAT
         WRITELN;
         WRITE ('Quanti punti?');
         READLN(N)
      UNTIL N <= MAX;
      FOR I := 1 TO N DO
         X[I] := 100 * RANDOM(0);
      PRINT;
      WRITE(CHR(7));
      SORT(X, N) (* numeri random *);
      WRITE(CHR(7));
      PRINT;
```

Figura 6.1 — Una procedura di ordinamento a bolla con programma principale (segue).

```
      WRITELN(' random');
      WRITE(CHR(7));
      SORT(X, N) (* numeri ordinati *);
      WRITE(CHR(7));
      PRINT;
      WRITELN ('ordinato');
      FOR I := 1 TO N DO
         X[I] := N + 1 - I;
      WRITE(CHR(7));
      SORT(X, N) (* numeri inversi *);
      WRITE(CHR(7));
      PRINT;
      WRITELN ('inversi');
   UNTIL N < 5
END.
```

Figura 6.1 — Una procedura di ordinamento a bolla con programma principale (segue).

Studieremo ora una routine di ordinamento a bolla leggermente più efficiente, come capirete meglio in seguito.

Aggiunta di una routine separata di scambio

Quando un'operazione deve essere eseguita più di una volta, è opportuno usare una procedura. Più avanti inseriremo nel programma di interpolazione del capitolo precedente una procedura di ordinamento, in cui sarà necessario scambiare un dato di una matrice Y ogni volta che scambiamo il dato corrispondente nella matrice X; la parte centrale della routine di ordinamento diventerà allora:

```
BEGIN
  HOLD : = A[I];
  A[I] : = A[J]
  A[J] : = HOLD;
  HOLD : = B[I];
  B[I] : = B[J];
  B[J] : = HOLD
  END;
```

In questo modo si effettua prima lo scambio tra due elementi di un array, e poi lo scambio tra quelli corrispondenti dell'altro array. Come nel programma di Figura 6.1, l'operazione di scambio richiede una terza variabile che si chiama HOLD.

Invece che ripetere le istruzioni di scambio ogni volta che è necessario, è più elegante creare una procedura, come vedremo nella seconda versione della routine di ordinamento a bolla.

PROCEDURA PASCAL: ORDINAMENTO A BOLLA CON SCAMBIO

Nella Figura 6.2 c'è una nuova procedura di ordinamento, che contiene a sua volta una procedura separata di nome SWAP per effettuare lo scambio tra due elementi. Aggiungetela al vostro primo programma di ordinamento. Il codice sorgente di questa nuova versione è semplice da capire; inoltre la routine può essere facilmente modificata per ordinare due o tre array, A, B e C. Si possono aggiungere le istruzioni:

```
SWAP (B[I], B[J]);
SWAP (C[I], C[J]);
```

immediatamente dopo la prima chiamata alla procedura SWAP. La nuova versione è più rapida della prima, quando l'insieme dei dati è già ordinato; negli altri casi è più lenta.

Esecuzione della nuova versione

Sostituite la nuova versione della procedura SORT al posto dell'originale. Se la prima versione della SORT è su un file separato su disco, fate una seconda copia. Modificate la seconda versione secondo il listing di Figura 6.2 e datele un nome come SORT.PAS, o comunque come quello che avete usato nella direttiva INCLUDE. Conservate la versione originale su un file con un altro nome. Se la procedura SORT fa parte del vostro programma principale, fate una copia dell'intero programma, poi modificate la SORT secondo il listing di Figura 6.2. Compilate ed eseguite la nuova versione; i risultati dovrebbero essere gli stessi.

Notate che la SWAP è una procedura locale della SORT, e non può essere usata da altre procedure esterne alla SORT; se, in seguito, sarà necessaria per altre procedure, bisognerà renderla globale, separandola dal corpo della SORT.

Adesso studieremo una routine di ordinamento più sofisticata e un po' più difficile.

```
PROCEDURE (* bubble *) SORT(VAR A: ARY; N: INTEGER);

(* Adottato da ´Introduction to Pascal,´
      R. Zaks, Sybex, 1980 *)

VAR
   NO_CHANGE: BOOLEAN;
   J: INTEGER;

PROCEDURE SWAP(VAR P, Q: REAL);

VAR
   HOLD: REAL;

BEGIN
   HOLD := P;
   P := Q;
   Q := HOLD
END (* swap *);

BEGIN (* procedure sort *)
   REPEAT
      NO_CHANGE := TRUE;
      FOR J := 1 TO N - 1 DO
         BEGIN
            IF A[J] > A[J+1] THEN
               BEGIN
                  SWAP(A[J], A[J+1]);
                  NO_CHANGE := FALSE
               END
         END (* FOR *)
   UNTIL NO_CHANGE
END (* procedure sort *);
```

Figura 6.2 — Una variante dell'ordinamento a bolla, con procedura separata SWAP.

ORDINAMENTO DI SHELL

Il maggior difetto dell'ordinamento a bolla è che spesso esegue più confronti e scambi di quanto non sia necessario; può capitare che un elemento sia spostato da un capo all'altro dell'array, e, poco dopo, rimesso al posto di prima.

L'ordinamento di Shell-Metzner è, in genere, più efficiente; all'inizio i confronti vengono fatti tra elementi distanti, ad esempio tra il primo e quello di mezzo. Per insiemi con meno di una dozzina di elementi il metodo a bolla e quello di Shell-Metzner impiegano tempi confrontabili, mentre all'aumentare del numero di elementi quello di Shell diventa sensibilmente più veloce, soprattutto per insiemi con più di 50 elementi.

Con il programma TSTSORT della Figura 6.1 possiamo valutare la differenza di velocità tra i due metodi.

PROCEDURA PASCAL: ORDINAMENTO DI SHELL-METZNER

Modificate la vostra routine di ordinamento secondo il listing della Figura 6.3. Notate che la procedura SWAP è la stessa di prima. Eseguite il programma e confrontate il tempo di ordinamento con quello del metodo a bolla; vedrete che è minore, e anche che il programma è più veloce su dati ordinati.

```
PROCEDURE (* shell *) SORT(VAR A: ARY; N: INTEGER);

  (* ordinamento di Shell-Metzner *)
  (* Adattato da 'Programming in Pascal',
     P. Grogono, Addison—Wesley, 1980 *)

VAR
  DONE: BOOLEAN;
  JUMP, I, J: INTEGER;
```

Figura 6.3 — Una procedura di ordinamento di Shell.

```
PROCEDURE SWAP(VAR P, Q: REAL);

VAR
   HOLD: REAL;

BEGIN
   HOLD := P;
   P := Q;
   Q := HOLD
END (* swap *);


BEGIN
   JUMP := N;
   WHILE JUMP > 1 DO
      BEGIN
         JUMP := JUMP DIV 2;
         REPEAT
            DONE := TRUE;
            FOR J := 1 TO N - JUMP DO
               BEGIN
                  I := J + JUMP;
                  IF A[J] > A[I] THEN

                     BEGIN
                        SWAP(A[J], A[I]);
                        DONE := FALSE
                     END (* IF *)
               END (* FOR *)
         UNTIL DONE
      END (* WHILE *)
END (* sort *);
```

Figura 6.3 — Una procedura di ordinamento di Shell (segue).

Nella discussione dell'ultimo e più complesso metodo di ordinamento, introdurremo il concetto di ricorsività, e presenteremo una versione ricorsiva e una non ricorsiva del programma di ordinamento.

ORDINAMENTO VELOCE

Abbiamo visto che la tecnica di ordinamento a bolla è facile da programmare e da capire, mentre quella di Shell è un po' più complicata ma più veloce. Entrambi gli algoritmi possono essere codificati senza difficoltà in linguaggi evoluti come il BASIC e il FORTRAN. Il terzo algoritmo che considereremo si chiama "ordinamento veloce", ed è ancora più complesso dell'ultimo che abbiamo visto; in genere è più veloce sia di quello a bolla che di quello di Shell, a meno che i dati siano ordinati o quasi; in questo caso è molto più veloce quello di Shell, mentre l'ordinamento veloce impiega circa lo stesso tempo per insiemi di dati ordinati e disordinati.

PROCEDURA PASCAL: UNA ROUTINE RICORSIVA DI ORDINAMENTO VELOCE

La prima versione dell'ordinamento veloce è ricorsiva, cioè contiene una procedura che richiama se stessa; uno svantaggio di questo algoritmo è che non può essere codificato direttamente in BASIC o in FORTRAN proprio a causa della sua ricorsività.

L'ordinamento a bolla comincia col confrontare elementi vicini; quello di Shell confronta il primo elemento con quello di mezzo; l'ordinamento veloce confronta gli elementi che si trovano agli estremi opposti dell'array, in modo che il primo scambio può avvenire tra elementi molto distanti.

La procedura PARTIT viene chiamata ripetutamente per operare su diverse parti dell'array: divide l'array a metà e sposta gli elementi in modo da avere tutti quelli a sinistra più piccoli di quelli a destra; le due nuove parti sono nuovamente divise a metà e il procedimento viene ripetuto. La suddivisione continua fino ad avere diversi insiemi di un solo elemento; a questo punto evidentemente l'array è ordinato.

Vi conviene conservare entrambe le versioni dell'ordinamento a Shell e di quello veloce (Figura 6.4); in genere sarà più conveniente usare quest'ultimo, ma se vi capita di ordinare insiemi già quasi ordinati, vi converrà quello di Shell.

Nel prossimo paragrafo studieremo come convertire questo ordinamento veloce in una procedura non ricorsiva. Una procedura ricorsiva fa una copia di se stessa e di ogni parametro non variabile ad ogni livello di ricorsività; d'altra parte l'insieme ordinato con la versione ricorsiva dell'ordinamento veloce di Figura 6.4 era dichiarato come parametro variabile nelle procedure QSORT e PARTIT, e quindi non viene mai

duplicato. Questo è il motivo per cui è possibile scrivere una versione non ricorsiva della procedura. I parametri scalari, però, che costituiscono gli indici dei sottoinsiemi, sono unici, quindi sono duplicati ad ogni chiamata ricorsiva; dunque sarà necessario fare copie separate di questi parametri nella versione non ricorsiva.

```
PROCEDURE (* quick *) SORT(VAR X: ARY; N: INTEGER);
(* una routine di ordinamento ricorsiva *)
(* Adattato da 'The design of Well –
      Structured and Correct Programs,'
      S. Alagic, Springer–Verlag, 1978 *)
PROCEDURE QSORT(VAR X: ARY; M,N: INTEGER);

VAR
   I, J: INTEGER;
PROCEDURE PARTIT(VAR A: ARY; VAR I,J: INTEGER;
                              LEFT, RIGHT: INTEGER);
VAR
   PIVOT: REAL;

PROCEDURE SWAP(VAR P, Q: REAL);
VAR
   HOLD: REAL;

BEGIN
   HOLD := P;
   P := Q;
   Q := HOLD
END (* swap *);
```

Figura 6.4 – Una versione ricorsiva dell'ordinamento veloce.

```
BEGIN
   PIVOT := A[(LEFT + RIGHT) DIV 2];
   I := LEFT;
   J := RIGHT;
   WHILE I <= J DO
      BEGIN
         WHILE A[I] < PIVOT DO
            I := I + 1;
         WHILE PIVOT < A[J] DO
            J := J - 1;
         IF I <= J THEN
            BEGIN
               SWAP(A[I], A[J]);
               I := I + 1;
               J := J - 1
            END
      END (* WHILE *)
END (* partit *);

BEGIN (* qsort *)
   IF M < N THEN
      BEGIN
         PARTIT(X, I, J, M, N) (* divide in due *);
         QSORT(X, M, J) (* ordina parte sinistra *);
         QSORT(X, I, N) (* ordina parte destra *)
      END
END (* qsort *);

BEGIN (* sort *)
   QSORT(X, 1, N)
END (* sort *);
```

Figura 6.4 — Una versione ricorsiva dell'ordinamento veloce (segue).

PROCEDURA PASCAL: UNA ROUTINE NON RICORSIVA DI ORDINAMENTO VELOCE

La Figura 6.5 mostra una versione non ricorsiva dell'ordinamento veloce; il tempo di esecuzione è un po' più lungo di quella ricorsiva, che resta quindi più conveniente se si programma in Pascal; la versione non ricorsiva è invece necessaria in BASIC e in FORTRAN, linguaggi che non consentono ricorsività. I parametri di destra e di sinistra per le chiamate simulate sono contenuti negli array interi LEFT e RIGHT.

Il pivot è inizialmente l'ultimo elemento dell'array; questa scelta è assolutamente inadeguata se l'array è già ordinato, in ordine crescente o decrescente. Perciò l'elemento pivot è confrontato con altri due, il primo e quello di mezzo; l'elemento di mezzo di questi tre, quindi nè il più piccolo nè il più grande, viene scelto come pivot. Questa scelta accelera di molto il processo di ordinamento quando l'array è già ordinato in un senso o nell'altro.

In questa versione, ad ogni suddivisione viene ordinato il più piccolo dei due sottoinsiemi prima del più grande, minimizzando così lo spazio per memorizzare gli indici non ordinati.

Torniamo ora al nostro programma di interpolazione, che è stato il punto di partenza per la discussione delle routine di ordinamento.

AGGIUNTA DELLA ROUTINE DI ORDINAMENTO NEL PROGRAMMA DI INTERPOLAZIONE DI UNA CURVA

Il passo successivo consisterà nell'inserire una delle routine di ordinamento nel programma di interpolazione del Capitolo 5. L'ordinamento veloce ricorsivo è probabilmente il migliore, ma potete anche scegliere quello di Shell. Anche ora tenete una copia della versione precedente del programma, in modo da avere un punto di partenza sicuro nell'eventualità che la nuova versione si complichi al di là del previsto.

Occorre fare due modifiche alla routine di ordinamento, in modo che l'array Y sia ordinato insieme a X, e precisamente l'array Y deve essere aggiunto alla lista dei parametri nell'intestazione, e va aggiunta un'altra chiamata alla procedura SWAP.

Se scegliete l'ordinamento veloce, modificate l'intestazione nel modo seguente:

```
PROCEDURE (*QUICK*) SORT
          (VAR X, Y : ARY;
               N : INTEGER);
```

```
PROCEDURE (* quick *) SORT(VAR X: ARY; N: INTEGER);

(* una routine di ordinamento rapido non ricorsiva *)
(* Adattata da 'Software Tools',
      B. Kernighan, Addison—Wesley, 1976 *)

VAR
   LEFT, RIGHT: ARRAY[1..20] OF INTEGER;
   I, J, SP,MID: INTEGER;
   PIVOT: REAL;

PROCEDURE SWAP( VAR P, Q: REAL);

VAR
   HOLD: REAL;

BEGIN
   HOLD := P;
   P := Q;
   Q := HOLD
END;

BEGIN
   LEFT[1] := 1;
   RIGHT[1] := N;
   SP := 1;
   WHILE SP > 0 DO
      BEGIN
         IF LEFT[SP] >= RIGHT[SP] THEN SP := SP — 1
         ELSE
```

Figura 6.5 — Una versione non ricorsiva dell'ordinamento veloce.

```
BEGIN
   I := LEFT[SP];
   J := RIGHT[SP];
   PIVOT := X[J];
   MID := (I + J) DIV 2;
   IF (J − I) > 5 THEN
      IF  ((X[MID] < PIVOT) AND (X[MID] > X[I]))
         OR
         ((X[MID] > PIVOT) AND (X[MID] < X[I]))
            THEN SWAP( X[MID], X[J])
      ELSE
         IF  ((X[I] < X[MID]) AND (X[I] > PIVOT))
            OR ((X[I] > X[MID]) AND (X[I] < PIVOT))
               THEN SWAP( X[I], X[J]);
   PIVOT := X[J];
   WHILE  I < J DO
      BEGIN
         WHILE X[I] < PIVOT DO
            I := I + 1;
         J := J − 1;
         WHILE (I < J) AND (PIVOT < X[J]) DO
            J := J − 1;
         IF I < J THEN SWAP(X[I], X[J])
      END (* WHILE *);
      J := RIGHT[SP] (* pivot to i *);
      SWAP(X[I], X[J]);
      IF I − LEFT[SP] >= RIGHT[SP] − I THEN
         BEGIN (* pone prima la parte più breve *)
            LEFT[SP+1] := LEFT[SP];
            RIGHT[SP+1] := I − 1;
            LEFT[SP] := I + 1
         END
```

Figura 6.5 — Una versione non ricorsiva dell'ordinamento veloce. (segue)

```
                ELSE
                  BEGIN
                    LEFT[SP+1] := I + 1;
                    RIGHT[SP+1] := RIGHT[SP];
                    RIGHT[SP] := I - 1
                  END;
              SP := SP + 1 (* push stack *)
            END (* IF *)
        END (* WHILE *)
    END (* introduce nella stack *)
```

Figura 6.5 — Una versione non ricorsiva dell'ordinamento veloce. (segue)

Poi, circa alla fine della procedura PARTIT, aggiungete l'istruzione:

```
SWAP (Y[I], Y[J]);
```

subito dopo la chiamata:

```
SWAP (A[I], A[J]);
```

La routine di ordinamento può essere aggiunta al programma di interpolazione lineare in diversi modi; il più semplice è di inserirla subito dopo la procedura GET_DATA; per far ciò, inserite l'istruzione:

```
SORT(X, Y, N);
```

nel programma principale; se la routine è chiamata dopo il calcolo dell'interpolazione, avremo da lavorare su tre array: X, Y e Y_CALC. Perciò eseguiremo l'ordinamento subito dopo avere ottenuto i dati dalla procedura GET_DATA, inserendo la chiamata alla procedura SORT subito dopo il comando:

```
GET_DATA(X, Y, N)
```

alla fine del programma principale.

Le direttive INCLUDE e EXTERN per la procedura di Ordinamento

In alternativa, possiamo mettere la procedura di ordinamento in un file su disco separato, chiamato SORT.PAS, e richiamarlo con una direttiva:

```
(*$I SORT.PAS*)
```

Inserite questa linea appena prima della direttiva per la procedura PLOT. Questo approccio è molto più versatile; si possono provare facilmente tutte e quattro le routine di ordinamento senza modificare il programma di interpolazione. Naturalmente bisogna ricompilare l'insieme del programma e della routine. Supponiamo che le routine di ordinamento a bolla, di Shell, e le due di ordinamento veloce si chiamino rispettivamente:

SORTB.PAS
SORTS.PAS
SORTQ.PAS
SORTN.PAS

Potete cambiare il nome di ognuna di esse in SORT.PAS, e eseguire il programma principale.

Una terza possibilità è quella di mettere le routine di ordinamento in un file separato, e richiamarle con un'istruzione del tipo:

```
PROCEDURE SORT (VAR X, Y: ARY;
                    N : INTEGER);
EXTERN;
```

Eseguite il programma di interpolazione dopo aver aggiunto la routine di ordinamento; verificate che gli elementi dell'array X siano messi in ordine crescente, e che l'array Y sia ordinato contemporaneamente a X. La pendenza del grafico prodotto dovrebbe essere positiva.

SOMMARIO

Abbiamo visto le differenze tra tre comuni routine di ordinamento: quella a bolla, quella di Shell e quella veloce; sono tutte previste per operare su numeri reali, ma tenete presente che possono essere facilmente modificate per ordinare numeri interi, caratteri o stringhe di caratteri; basta ricordarsi di dichiarare la variabile HOLD dello stesso tipo degli elementi dell'array X che deve essere ordinato.

Ora che il programma di interpolazione di una curva è in grado di trattare dati sperimentali reali, possiamo usarlo con equazioni più complesse; questo è l'argomento del Capitolo 7.

CAPITOLO 7

INTERPOLAZIONE DI UNA CURVA CON IL METODO DEI MINIMI QUADRATI

INTRODUZIONE

In questo capitolo svilupperemo diversi programmi di interpolazione di una curva con il metodo dei minimi quadrati. Ci proponiamo di generalizzare il programma che abbiamo già visto, in modo che diventi uno strumento più utile e più adeguato alla realtà, in una vasta gamma di situazioni sperimentali. Finora ci siamo limitati all'equazione di una retta; il primo programma di questo capitolo è fatto per una parabola, cioè una curva descritta da un'equazione polinomiale di secondo grado. In seguito ci occuperemo di equazioni, polinomiali e no, di grado superiore, in cui i termini contenenti l'incognita sono lineari.

Useremo un vettore e una matrice di dati; quando il programma è in esecuzione, dobbiamo inserire da tastiera il grado dell'equazione polinomiale. Infine realizzeremo un programma di interpolazione per dati sperimentali; tra le equazioni risolte ci sono quelle della capacità termica e della pressione di un vapore. Inoltre, nel programma per un'equazione di stato a tre variabili, cercheremo soluzioni per equazioni a coefficienti non lineari.

INTERPOLAZIONE DI UNA PARABOLA

Nel Capitolo 5 abbiamo scritto un programma di interpolazione con i minimi quadrati, per calcolare i coefficienti A e B dell'espressione:

$$y = Ax + B \tag{1}$$

La (1) è l'equazione più comune di interpolazione di una curva, ma in certi casi è

necessario usarne una diversa. In questo capitolo vedremo l'applicazione del metodo dei minimi quadrati ad altre espressioni, altrettanto comuni.

Poniamo come condizione che le equazioni siano lineari nei coefficienti delle incognite, come:

$$y = A + \frac{A}{x} + Cz$$

$$y = A + B/x + Cz$$

$$\ln y = \frac{Ax}{z} + Be^x$$

dove A, B e C sono lineari. Non considereremo invece equazioni del tipo:

$$y = A + Be^{Cx} \tag{2}$$

perchè C non è lineare.

Lo sviluppo di un programma di interpolazione per una delle equazioni precedenti è analogo a quello usato nel Capitolo 5. Consideriamo ad esempio l'equazione di una parabola, polinomiale di secondo grado:

$$y = A + Bx + Cx^2$$

Definiamo i residui r:

$$r = A + Bx + Cx^2 - y$$

Eleviamo al quadrato i residui e sommiamo i risultati; come nel Capitolo 5 vogliamo minimizzare questa quantità, calcolando la derivata rispetto ad ognuna delle variabili (A, B e C, in questo caso), e ponendola uguale a zero. Nel caso della parabola, otterremo tre equazioni, una per ognuna delle tre variabili:

$$An + B\Sigma x + C\Sigma x^2 = \Sigma y \tag{3}$$

$$A\Sigma x + B\Sigma x^2 + C\Sigma x^3 = \Sigma xy \tag{4}$$

$$A\Sigma x^3 + B\Sigma x^3 + C\Sigma x^4 = \Sigma x^2 y \tag{5}$$

PROGRAMMA PASCAL: INTERPOLAZIONE DI UNA PARABOLA CON I MINIMI QUADRATI

Questa soluzione si ottiene risolvendo il sistema di equazioni costituito dalle 3, 4 e 5., col programma della figura 7.1, che usa la regola di Cramer; si devono quindi calcolare i determinanti di quattro matrici 3 per 3, e per far ciò si usa la funzione DETEr del Capitolo 4 (Figura 4.2).

```
PROGRAM LEAST1(OUTPUT);
  (* 26 Dic 80 *)

(* Programma pascal che esegue una *)
(* interpolazione lineare coi minimi quadrati *)
(* con l'uso di una curva parabolica *)
(* è necessaria una procedura separata PLOT *)

CONST
  MAXR = 20;
  MAXC = 3;

TYPE
  ARY   = ARRAY[1..MAXR] OF REAL;
  ARYS  = ARRAY[1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXC, 1..MAXC] OF REAL;

VAR
  X, Y, Y_CALC : ARY;
  COEF         : ARYS;
  NROW, NCOL   : INTEGER;
  CORREL_COEF: REAL;

PROCEDURE GET_DATA(VAR X, Y   : ARY;
                   VAR NROW : INTEGER);

(* riceve i valori di nrow e delle matrici x, y *)

VAR
  I: INTEGER;
```

Figura 7.1 — **Interpolazione parabolica con i minimi quadrati.**

```
BEGIN
   NROW := 9;
   FOR I := 1 TO NROW DO X[I] := I;
   Y[1] := 2.07; Y[2] := 8.6;
   Y[3] := 14.42; Y[4] := 15.80;
   Y[5] := 18.92; Y[6] := 17.96;
   Y[7] := 12.98; Y[8] := 6.45;
   Y[9] := 0.27
END (* procedure get_data *);

PROCEDURE WRITE_DATA;

(* stampa i risultati *)

VAR
   I: INTEGER;


BEGIN
   WRITELN;
   WRITELN('        I        X        Y        Y CALC');
   FOR I := 1 TO NROW DO
      WRITELN(I:3, X[I]:8:1, Y[I]:9:2, Y_CALC[I]:9:2);
   WRITELN; WRITELN ('Coefficienti');
   FOR I := 1 TO NCOL DO
      WRITELN( COEF[I]:8:4);
   WRITELN;
   WRITELN ('Il coefficiente di correlazione è',
                  CORREL_COEF:8:5)
END (* write_data *);
```

Figura 7.1 — Interpolazione parabolica con i minimi quadrati (segue).

```
PROCEDURE SOLVE( A : ARY2S;
                 Y : ARYS;
        VAR  COEF : ARYS;
             NROW : INTEGER;
        VAR ERROR : BOOLEAN);

VAR
  B    : ARY2S;
  I, J : INTEGER;
  DET  : REAL;

FUNCTION DETER(A: ARY2S): REAL;
(* calcola il determinante di una matrice 3-per-3 *)

BEGIN (* function deter *)
  DETER := A[1,1] * (A[2,2] * A[3,3] - A[3,2] * A[2,3])
           - A[1,2] * (A[2,1] * A[3,3] - A[3,1] * A[2,3])
           + A[1,3] * (A[2,1] * A[3,2] - A[3,1] * A[2,2])
END (* function deter *);

PROCEDURE SETUP(VAR B
                VAR COEF : ARYS;
                    J    : INTEGER);
VAR
  I: INTEGER;

BEGIN (* setup *)
  FOR I := 1 TO NROW DO
    BEGIN
      B[I,J] := Y[I];
      IF J > 1 THEN B[I,J-1] := A[I,J-1]
    END;
  COEF[J] := DETER(B)/DET
END (* setup *);
```

Figura 7.1 — Interpolazione parabolica con i minimi quadrati (segue).

```
BEGIN (* procedure solve *)
  ERROR := FALSE;
  FOR I := 1 TO NROW DO
    FOR J := 1 TO NROW DO
      B[I,J] := A[I,J];
  DET := DETER(B);
  IF DET = 0.0 THEN
    BEGIN
      ERROR := TRUE;
      WRITELN ('ERRORE: matrice singolare')
    END
  ELSE
    BEGIN
      SETUP( B, COEF, 1);
      SETUP( B, COEF, 2);
      SETUP( B, COEF, 3)
    END (* else *)
END (* procedure solve *);

PROCEDURE LINFIT(X, Y: ARY;
        VAR Y_CALC  : ARY;
        VAR COEF    : ARYS;
            NROW    : INTEGER;
        VAR NCOL    : INTEGER);

(* i minimi quadrati approssimano a una parabola *)
(* nrow gruppi di coppie di punti x e y *)

VAR
  A : ARY2S;
  G : ARYS;
  I : INTEGER;
```

Figura 7.1 — Interpolazione parabolica con i minimi quadrati (segue).

```
    ERROR: BOOLEAN;
    SUM_X, SUM_Y, SUM_XY, SUM_X2,
    SUM_Y2, XI, YI, SXY ,SXX, SYY,
    SUM_X3, SUM_X4, SUM_2Y, DENOM,
    SRS, X2: REAL;

BEGIN (* linfit *)
    NCOL := 3 (* termini polinomiali *)
    SUM_X := 0;
    SUM_Y := 0;
    SUM_XY := 0;
    SUM_X2 := 0;
    SUM_Y2 := 0;
    SUM_X3 := 0;
    SUM_X4 := 0;
    SUM_2Y := 0;
    FOR I := 1 TO NROW DO


BEGIN
    XI := X[I];
    YI := Y[I];
    X2 := XI * XI;
    SUM_X := SUM_X + XI;
    SUM_Y := SUM_Y + YI;
    SUM_XY := SUM_XY + XI * YI;
    SUM_X2 := SUM_X2 + X2;
    SUM_Y2 := SUM_Y2 + YI * YI;
    SUM_X3 := SUM_X3 + XI * X2;
    SUM_X4 := SUM_X4 + X2 * X2;
    SUM_2Y := SUM_2Y + X2 * YI
END;
```

Figura 7.1 — Interpolazione parabolica con i minimi quadrati (segue).

```
      A[1,1] := NROW;
      A[2,1] := SUM_X; A[1,2] := SUM_X;
      A[3,1] := SUM_X2; A[1,3] := SUM_X2;
      A[2,2] := SUM_X2; A[3,2] := SUM_X3;
      A[2,3] := SUM_X3; A[3,3] := SUM_X4;
      G[1] := SUM_Y;
      G[2] := SUM_XY;
      G[3] := SUM_2Y;
      SOLVE( A, G, COEF, NCOL, ERROR);
      SRS := 0.0;
      FOR I := 1 TO NROW DO

      BEGIN
         Y_CALC[I] :=
            COEF[1] + COEF[2] * X[I] + COEF[3] * SQR(X[I]);
         SRS := SRS + SQR(Y[I] - Y_CALC[I])
      END;
   CORREL_COEF :=
   SQRT(1.0 - SRS/(SUM_Y2 -SQR(SUM_Y)/NROW))
END (* linfit *);

(*
PROCEDURE plot
            (x, y, y_calc: ary; nrow: integer);
extern; *)
(*$F PLOT.PAS *) (* get procedure PLOT *)
BEGIN (* programma principale *)
   GET_DATA(X, Y, NROW);
   LINFIT(X, Y, Y_CALC, COEF, NROW, NCOL);
   WRITE_DATA;
   PLOT(X, Y, Y_CALC, NROW)
END.
```

Figura 7.1 — Interpolazione parabolica con i minimi quadrati (segue).

Nei programmi dei capitoli precedenti i coefficienti erano A, B e C; in questo capitolo useremo il vettore COEF, in modo che A corrisponda al primo elemento del vettore, COEF(1), B a COEF(2) e C a COEF(3).

Notate che il programma principale chiama la procedura PLOT, che abbiamo usato nei precedenti capitoli; il suo listing non compare però nel programma, e viene richiamata con una direttiva INCLUDE. Se il vostro Pascal non ha questa possibilità, dovete inserire nel programma una copia della PLOT.

La PLOT richiede che l'array della variabile indipendente sia in ordine crescente o decrescente, e i dati forniti dalla GET_DATA sono così; se, in un secondo tempo volete usare dati non ordinati, dovete aggiungere una delle procedure di ordinamento del Capitolo 6.

Esecuzione del programma

Scrivete il programma ed eseguitelo; i risultati si presenteranno come in Figura 7.2. Il coefficiente di correlazione è prossimo a 1, il che significa che l'equazione della parabola può dare una buona interpolazione. L'equazione risultante è:

$$y = -7.827 + 10.59x - 1.083x^2$$

Ora consideriamo equazioni polinomiali di grado superiore a 2 e equazioni non polinomiali; vedremo che l'approccio usato per determinare i coefficienti è inadeguato per equazioni più complesse, e bisogna trovarne un altro.

INTERPOLAZIONE DI CURVE PER ALTRE EQUAZIONI

Se abbiamo un'equazione polinomiale di grado superiore a 2, ci saranno altri coefficienti, e quindi altre equazioni da risolvere simultaneamente;

Le equazioni 3, 4 e 5 possono essere facilmente estese a questo caso. Per esempio, per trovare i coefficienti dell'equazione cubica:

$$y = A + Bx + Cx^2 + Dx^3$$

i residui sono:

$$r = A + B\,x + Cx^2 + Dx^3 - y$$

I	X	Y	Y CALC
1	1.0	2.07	1.68
2	2.0	8.60	9.02
3	3.0	14.42	14.20
4	4.0	15.80	17.21
5	5.0	18.92	18.05
6	6.0	17.96	16.73
7	7.0	12.98	13.24
8	8.0	6.45	7.58
9	9.0	0.27	-0.24

Coefficienti
-7.8267
10.5901
-1.0830

Il coefficiente di correlazione è 0.99155

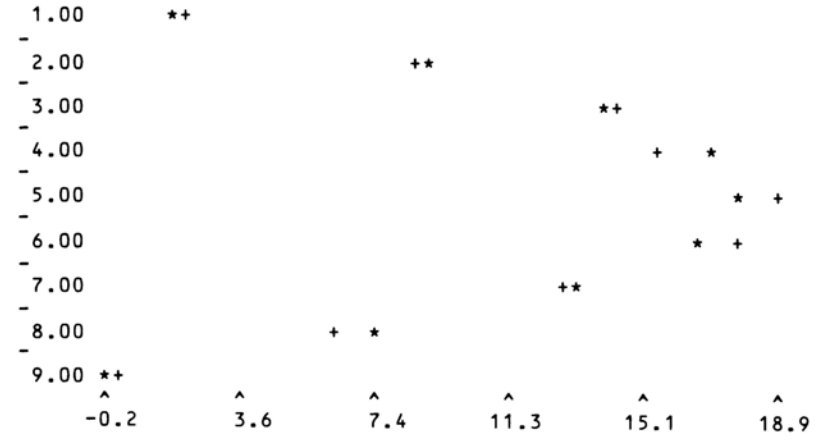


Figura 7.2 — Risultati del programma di interpolazione parabolica.

Si elevano al quadrato i residui e si sommano i risultati; si esegue la derivata rispetto ad ogni variabile a, B, C e si ottiene un sistema di quattro equazioni:

$$An + B\Sigma x + C\Sigma x^2 + D\Sigma x^3 = \Sigma y$$
$$A\Sigma x + B\Sigma x^2 + C\Sigma x^3 + D\Sigma x^4 = \Sigma xy$$
$$A\Sigma x^2 + B\Sigma x^3 + C\Sigma x^4 + D\Sigma x^5 = \Sigma x^2 y$$
$$A\Sigma x^3 + B\Sigma x^4 + C\Sigma x^5 + D\Sigma x^6 = \Sigma x^3 y$$

Le espressioni non polinomiali sono trattate analogamente. Per esempio, un'equazione del tipo:

$$y = A + Bf(x) + Cg(x)$$

dove f(x) e g(x) rappresentano delle funzioni di x, ha residui:

$$r = A + Bf(x) + Cg(x) - y$$

Questi vengono elevati al quadrato e poi sommati. La derivata rispetto alle tre variabili dà le seguenti tre equazioni:

$$An + B\Sigma f(x) + C\Sigma g(x) = \Sigma y$$
$$A\Sigma f(x) + B\Sigma f(x^2) + C\Sigma f(x)g(x) = \Sigma f(x)y$$
$$A\Sigma g(x) + B\Sigma f(x)g(x) + C\Sigma g(x)^2 = \Sigma g(x)y$$

Questo metodo è corretto, ma molto laborioso; ogni nuova equazione richiede ulteriori modifiche. Per esempio, se l'equazione è:

$$y = A + Bx + C/x^2$$

saranno necessarie nel programma istruzioni del tipo:

```
SUM_X3 : = SUM_X3 + 1/XI;
SUM_X4 : = SUM_X4 + 1/(XI*XI);
```

per calcolare le somme.

Una soluzione diretta

Per determinare i coefficienti dell'equazione è meglio organizzare i dati in una matrice e un vettore, che danno poi luogo ad un sistema di equazioni che si risolvono nel solito modo. Supponiamo di volere un'interpolazione lineare con l'equazione

$$y = A + Bx$$

per cinque insiemi di dati x-y. Il vettore dei dati in questo caso contiene i valori di y, e la matrice è del tipo:

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ 1 & x_5 \end{bmatrix}$$

Ogni riga della matrice dei dati corrisponde a un punto, ogni colonna a un termine dell'equazione; quindi la matrice ha cinque righe e due colonne. La prima colonna contiene solo 1, poichè questo è la funzione di x (x^0) nel primo termine dell'equazione; la seconda contiene i valori di x, poichè questa è la funzione di x nel secondo termine dell'equazione.

La matrice dei dati per un'interpolazione parabolica avrà tre colonne; le prime due saranno le stesse della retta, la terza contiene i quadrati dei valori di x. Invece, per un'equazione del tipo:

$$\ln p = A + \frac{B}{t} + C \ln t$$

la prima colonna contiene 1, la seconda il reciproco dei dati, e la terza il logaritmo dei dati. Il vettore dei dati contiene il logaritmo di p.

La matrice rettangolare è trasformata in quadrata con una semplice operazione: la trasposta della matrice viene moltiplicata per la matrice stessa per produrre la matrice dei coefficienti. Nel caso di interpolazione con la retta di cinque insiemi di dati x-y, l'operazione è:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix} \quad \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ 1 & x_5 \end{bmatrix}$$

Il risultato è una matrice 2 per 2 che contiene le necessarie sommatorie di x:

$$\begin{bmatrix} n & \Sigma x \\ \Sigma x & \Sigma x^2 \end{bmatrix}$$

Il prodotto del vettore dei dati (considerato come vettore riga) e la matrice dei dati:

$$\begin{bmatrix} y_1 & y_2 & y_3 & y_4 & y_5 \end{bmatrix} \quad \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ 1 & x_5 \end{bmatrix}$$

dà come risultato il vettore dei termini noti di lunghezza 2:

$$\begin{bmatrix} \Sigma y & \Sigma xy \end{bmatrix}$$

Vediamo ora un programma in Pascal che realizza questo procedimento, e che contiene una procedura generale per calcolare l'errore standard. Come abbiamo anticipato nel Capitolo 4, il metodo di Gauss-Jordan di risoluzione di sistemi di equazioni è un ottimo strumento da usare in questo caso, perchè fornisce l'inversa della matrice dei coefficienti.

PROGRAMMA PASCAL: L'APPROCCIO PER MATRICI ALL'INTERPOLAZIONE DI CURVE

Il programma della Figura 7.3 usa questo approccio per matrici. La matrice e il vettore dei dati sono creati all'inizio della procedura LINFIT; poi, la procedura SQUARE converte la matrice X e il vettore Y nella matrice quadrata dei coefficienti A, e nel vettore dei termini noti **g**. Le operazioni sono le seguenti:

$$X^T X = A$$

e

$$\mathbf{y}X = \mathbf{g}$$

Il prodotto del vettore dei dati per la matrice dà per risultato il vettore g. Il vettore contenente la soluzione:

$$A^{-1}\mathbf{g} = \mathbf{b}$$

si può ottenere usando una qualsiasi delle routine del Capitolo 4 per la risoluzione di sistemi di equazioni.

Il programma di interpolazione lineare del Capitolo 5 presentava gli errori standard insieme con i valori corrispondenti della funzione di approssimazione. Gli errori standard si ottenevano dall'errore standard della stima e le sommatorie di x e di x^2. Qui useremo una tecnica più generale.

Gli errori standard si ottengono facilmente dall'inversa della matrice dei coefficienti; il valore corrispondente al termine j-esimo della funzione di approssimazione è il prodotto dell'errore standard della stima (SEE) per la radice quadrata del termine j-esimo della diagonale principale dell'inversa:

$$\begin{bmatrix} a_{11} & & \\ & a_{22} & \\ & & a_{33} \end{bmatrix}^{-1}$$

L'espressione Pascal è:

```
SIG[I] := SEE*SQRT(A[I,I])
```

Fra i diversi metodi che abbiamo visto finora per la soluzione di sistemi di equazioni lineari, solo quello di Gauss-Jordan genera l'inversa della matrice dei coefficienti. Poichè questa ci serve per determinare gli errori degli elementi del vettore soluzione, nel resto del capitolo useremo questo metodo per i programmi di interpolazione.

Gli errori standard sui coefficienti si possono usare per determinare l'intervallo di affidabilità per i coefficienti stessi. Inoltre, gli errori standard ci possono segnalare la possibilità di equazioni mal condizionate. Abbiamo visto le matrici mal condizionate nel Capitolo 4; è più probabile incontrarle nella soluzione di sistemi di equazioni che nell'interpolazione di curve, ma affronteremo ugualmente il problema.

Gli errori standard si ricavano dalla radice quadrata degli elementi della diagonale principale della matrice inversa. Notevoli differenze in questi elementi segnalano la possibilità di equazioni mal condizionate; perciò, se i quadrati degli errori differiscono per diversi ordini di grandezza può darsi che le equazioni siano mal condizionate. L'ultimo programma nel capitolo ne dà un esempio.

Il programma della Figura 7.3 è simile a quello della Figura 7.1, ma la soluzione dell'equazione parabolica si trova con il metodo più generale delle matrici; il programma usa tre routine già viste. La procedura di moltiplicazione tra matrici del Capitolo 3 compare nel listing, mentre vengono richiamate come file separati la procedura GAUSSJ di Gauss-Jordan del Capitolo 4 e la PLOT.

```
PROGRAM LEAST2(OUTPUT);
(* 26 Dic 80 *)
(* Programma pascal che esegue una *)
(* interpolazione lineare coi minimi quadrati *)
(* con la routine Gauss-Jordan *)
(*Sono necessarie le procedure:
          GAUSSJ, PLOT *)

CONST
  MAXR = 20; (* data punti *)
  MAXC = 4; (* termini polinomiali *)

TYPE
  ARY  = ARRAY[1..MAXR] OF REAL;
  ARYS = ARRAY[1..MAXC] OF REAL;
  ARY2 = ARRAY[1..MAXR, 1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXC, 1..MAXC] OF REAL;
```

Figura 7.3 — Interpolazione parabolica con i minimi quadrati con il metodo di eliminazione di Gauss-Jordan.

```
VAR
   X, Y, Y_CALC,
   RESID          : ARY;
   COEF, SIG      : ARYS;
   NROW, NCOL     : INTEGER;
   CORREL_COEF    : REAL;


PROCEDURE GET_DATA
            (VAR X      : ARY; (* variabile indipendente *)
             VAR Y      : ARY; (* variabile dipendente *)
             VAR NROW : INTEGER); (* lunghezza dei vettori *)

VAR
   I: INTEGER;
BEGIN
   NROW := 9;
   FOR I := 1 TO NROW DO
      X[I] := I;
   Y[1] := 2.07; Y[2] := 8.6;
   Y[3] := 14.42; Y[4] := 15.80;
   Y[5] := 18.92; Y[6] := 17.96;
   Y[7] := 12.98; Y[8] := 6.45;
   Y[9] := 0.27
END (* procedure get_data *);

PROCEDURE WRITE_DATA;
(* stampa i risultati *)

VAR
   I: INTEGER;
```

Figura 7.3 — Interpolazione parabolica con i minimi quadrati con il metodo di eliminazione di Gauss-Jordan (segue).

```
BEGIN
  WRITELN;
  WRITELN;
  WRITELN(' I   X    Y   Y CALC  RESID');
  FOR I := 1 TO NROW DO
     WRITELN(I: 3, X[I]: 8: 1, Y[I]: 9: 2,
        Y_CALC[I]: 9: 2, RESID[I]: 9: 2);
  WRITELN;
  WRITELN ('coefficienti errori');
  WRITELN(COEF[1], '  ', SIG[1], ' Termine costante');


   FOR I := 2 TO NCOL DO
      WRITELN(COEF[I], '  ', SIG[I]) (* altri termini *);
   WRITELN;
   WRITELN
      ('Il coefficiente di correlazione è', CORREL_COEF: 8: 5)
END (* write_data *);
PROCEDURE SQUARE(X : ARY2;
                 Y : ARY;
            VAR A : ARY2S;
            VAR G : ARYS;
      NROW,NCOL : INTEGER);
(* routine di moltiplicazione di matrici *)
(* A = traspone X volte X *)
(* G = Y volte X *)
VAR
  I, K, L: INTEGER;
BEGIN (* square *)
  FOR K := 1 TO NCOL DO
    BEGIN
      FOR L := 1 TO K DO
```

Figura 7.3 — Interpolazione parabolica con i minimi quadrati con il metodo di eliminazione di Gauss-Jordan (segue).

```
            BEGIN
              A[K,L] := 0.0;
              FOR I := 1 TO NROW DO
                BEGIN
                  A[K,L] := A[K,L] + X[I,L] * X[I,K];
                  IF K <> L THEN A[L,K] := A[K,L]
                END
            END (* ciclo L *);
          G[K] := 0.0;
          FOR I := 1 TO NROW DO
            G[K] := G[K] + Y[I] * X[I,K]
        END (* ciclo k *)
    END (* square *);

    (* PROCEDURE gaussj
       (VAR b     : ary2s;
            y     : arys;
        VAR coef  : arys;
            ncol  : integer;
        VAR error : boolean);
    extern; *)
    (*$F GAUSSJ.PAS *)

    PROCEDURE LINFIT(X,          (* variabile indipendente *)
                     Y : ARY; (* variabile dipendente *)
          VAR Y_CALC: ARY; (* var. dip. calcolata *)
          VAR RESID   : ARY; (* matrice dei resti *)
          VAR  COEF   : ARYS; (* coefficienti *)
          VAR  SIG    : ARYS; (* errori nei coefficienti *)
               NROW   : INTEGER; (* lunghezza dei vettori *)
          VAR  NCOL   : INTEGER); (* numero di termini *)
```

Figura 7.3 — Interpolazione parabolica con i minimi quadrati con il metodo di eliminazione di Gauss-Jordan (segue).

```
(* interpolazione lineare coi minimi quadrati di *)
(* nrow coppie di punti x e y *)
(* Sono necessarie le procedure separate:
   SQUARE — formazione della matrice dei coefficienti
   GAUSSJ — eliminazione di Gauss-Jordan *)

VAR
   XMATR  : ARY2; (* matrice dati *)
   A      : ARY2S; (* matrice coefficienti *)
   G      : ARYS; (* matrice costanti *)
   ERROR  : BOOLEAN;
   I, J, NM : INTEGER;
   XI, YI, YC, SRS, SEE,
   SUM_Y, SUM_Y2: REAL;


BEGIN (* procedure linfit *)
   NCOL := 3; (* numero di termini *)
   FOR I := 1 TO NROW DO
      BEGIN (* caricamento matrice X *)
         XI := X[I];
         XMATR[I, 1] := 1.0 (* prima colonna *)
         XMATR[I, 2] := XI (* seconda colonna *)
         XMATR[I, 3] := XI * XI (* terza colonna *)
      END;
   SQUARE(XMATR, Y, A, G, NROW, NCOL);
   GAUSSJ(A, G, COEF, NCOL, ERROR);
   SUM_Y := 0.0;
   SUM_Y2 := 0.0;
   SRS := 0.0;
   FOR I := 1 TO NROW DO
```

Figura 7.3 — Interpolazione parabolica con i minimi quadrati con il metodo di eliminazione di Gauss-Jordan (segue).

```
      BEGIN
         YI := Y[I];
         YC := 0.0;
         FOR J := 1 TO NCOL DO
            YC := YC + COEF[J] * XMATR[I, J];
         Y_CALC[I] := YC;
         RESID[I] := YC - YI;
         SRS := SRS + SQR(RESID[I]);
         SUM_Y := SUM_Y + YI;
         SUM_Y2 := SUM_Y2 + YI * YI
      END;
   CORREL_COEF :=
      SQRT(1.0 - SRS/(SUM_Y2 - SQR(SUM_Y)/NROW));
   IF NROW = NCOL THEN NM := 1
   ELSE NM := NROW - NCOL;
   SEE := SQRT(SRS/NM);
   FOR I := 1 TO NCOL DO (* errori nella soluzione *)
      SIG[I] := SEE * SQRT(A[I, I])
END (* linfit *);


(* PROCEDURE plot(x, y, z: ary; nrow: integer);
extern; *)
(*$F PLOT.PAS *)

BEGIN (* programma principale *)
   GET_DATA(X, Y, NROW);
   LINFIT(X, Y, Y_CALC, RESID, COEF, SIG, NROW, NCOL);
   WRITE_DATA;
   PLOT(X, Y, Y_CALC, NROW)
END.
```

Figura 7.3 — Interpolazione parabolica con i minimi quadrati con il metodo di eliminazione di Gauss-Jordan (segue).

Esecuzione del programma

Modificate il primo programma del capitolo, oppure createne un altro in base al listing di Figura 7.3; assicuratevi di avere disponibili le procedure GAUSSJ e PLOT su file esterni; in caso contrario, inserite queste procedure direttamente nel programma principale. Eseguite il programma e confrontate i risultati con la Figura 7.4; dovrebbero essere uguali a quelli del programma precedente, con l'unica differenza che nella nuova versione sono dati i residui e gli errori.

I	X	Y	Y CALC	RESID
1	1.0	2.07	1.68	-0.39
2	2.0	8.60	9.02	0.42
3	3.0	14.42	14.20	-0.22
4	4.0	15.80	17.21	1.41
5	5.0	18.92	18.05	-0.87
6	6.0	17.96	16.73	-1.23
7	7.0	12.98	13.24	0.26
8	8.0	6.45	7.58	1.13
9	9.0	0.27	-0.24	-0.51

coefficienti	errori	
-7.82657e0	1.29802e0	Termine costante
1.05900e1	5.96000e-1	
-1.08295e0	5.81267e-2	

Il coefficiente di correlazione è 0.99155

Figura 7.4 — Risultati della versione alternativa dell'interpolazione parabolica con i minimi quadrati.

Nel prossimo paragrafo, l'approccio per matrici ci consentirà di tentare l'interpolazione di un qualsiasi insieme di dati con equazioni polinomiali di diversi gradi. Per provare la potenza di questo strumento, eseguiremo il programma più volte sullo stesso insieme di dati; confrontando le curve ottenute sui grafici e i coefficienti di correlazione, determineremo il grado più opportuno per interpolare i nostri dati.

PROGRAMMA PASCAL: DETERMINAZIONE DEL GRADO DEL POLINOMIO

Un vantaggio della nuova versione del programma di interpolazione è la facilità con cui si possono cambiare il numero di righe, corrispondente al numero di punti, e

il numero delle colonne, corrispondente al numero di termini polinomiali nell'equazione di interpolazione. Nella terza versione, andremo ancora più in là nella generalizzazione del programma; inseriremo dalla console il grado dell'equazione polinomiale, che è naturalmente inferiore di 1 al numero di termini dell'equazione. Fate una copia del programma precedente e modificatelo secondo il listing della Figura 7.5. Dovete cambiare sia il programma principale che la procedura LINFIT.

```
PROGRAM LEAST3(INPUT, OUTPUT);
(* 26 Dic 80 *)
(* Programma pascal che esegue una *)
(* interpolazione lineare coi minimi quadrati *)
(* con la routine Gauss-Jordan *)
(* Sono necessarie le procedure separate:
          GAUSSJ, PLOT *)

CONST
  MAXR = 20; (* data punti *)
  MAXC = 4; (* termini polinomiali *)

TYPE
  ARY   = ARRAY[1..MAXR] OF REAL;
  ARYS  = ARRAY[1..MAXC] OF REAL;
  ARY2  = ARRAY[1..MAXR, 1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXC, 1..MAXC] OF REAL;

VAR
  X, Y, Y_CALC,
  RESID          : ARY;
  COEF, SIG      : ARYS;
  NROW, NCOL     : INTEGER;
  CORREL_COEF    : REAL;
  DONE           : BOOLEAN;
```

Figura 7.5 — Introduzione dell'ordine del polinomio dalla console.

```
PROCEDURE GET_DATA
            (VAR X      : ARY; (* variabile indipendente *)
             VAR Y      : ARY; (* variabile dipendente *)
             VAR NROW : INTEGER); (* lunghezza dei vettori *)

VAR
   I: INTEGER;


BEGIN
   NROW := 9;
   FOR I := 1 TO NROW DO
      X[I] := I;
   Y[1] := 2.07; Y[2] := 8.6;
   Y[3] := 14.42; Y[4] := 15.80;
   Y[5] := 18.92; Y[6] := 17.96;
   Y[7] := 12.98; Y[8] := 6.45;
   Y[9] := 0.27
END (* procedure get_data *);


PROCEDURE WRITE_DATA;
(* stampa i risultati *)


VAR
   I: INTEGER;

BEGIN
   WRITELN;
   WRITELN;
   WRITELN(' I   X    Y   Y CALC  RESID');
```

Figura 7.5 — Introduzione dell'ordine del polinomio dalla console (segue).

```
  FOR I := 1 TO NROW DO
     WRITELN(I: 3, X[I]: 8: 1, Y[I]: 9: 2,
        Y_CALC[I]: 9: 2, RESID[I]: 9: 2);
  WRITELN;
  WRITELN ('coefficienti errori');
  WRITELN(COEF[1], ' ', SIG[1], ' Termine costante');
  FOR I := 2 TO NCOL DO
     WRITELN
     (COEF[I], ' ', SIG[I]) (* altri termini *);
  WRITELN;
  WRITELN
     ('Il coefficiente di correlazione è', CORREL_COEF: 8: 5)
END (* write_data *);


(* PROCEDURE square(x : ary2;
                    y : ary;
                VAR a : ary2s;
                VAR g : arys;
            nrow,ncol : integer);
extern; *)
(*$F SQUARE.PAS *)

PROCEDURE gaussj
   (VAR b     : ary2s;
        y     : arys;
    VAR coef  : arys;
        ncol  : integer;
    VAR error : boolean);
extern; *)
(*$F GAUSSJ.PAS *)
```

Figura 7.5 — Introduzione dell'ordine del polinomio dalla console (segue).

```
PROCEDURE LINFIT(X,          (* variabile indipendente *)
                 Y : ARY; (* variabile dipendente *)
      VAR Y_CALC: ARY; (* var. dipendente calcolata *)
      VAR RESID   : ARY; (* matrice dei resti *)
      VAR  COEF   : ARYS; (* coefficienti *)
      VAR  SIG    : ARYS; (* errori nei coefficienti *)
           NROW   : INTEGER; (* lunghezza delle matrici *)
      VAR  NCOL   : INTEGER); (* numero dei termini *)
(* interpolazione lineare coi minimi quadrati di *)
(* nrow coppie di punti x e y *)
(* Sono necessarie le procedure separate:
   SQUARE — formazione matrice dei coefficienti
   GAUSSJ — eliminazione di Gauss-Jordan *)
VAR
   XMATR  : ARY2; (* matrice dei dati *)
   A      : ARY2S; (* matrice dei coefficienti *)
   G      : ARYS; (* vettore costanti *)
   ERROR  : BOOLEAN;
   I, J, NM : INTEGER;
   XI, YI, YC, SRS, SEE,
   SUM_Y, SUM_Y2: REAL;
BEGIN (* procedure linfit *)
   FOR I := 1 TO NROW DO
      BEGIN (* caricamento matrice X *)
         XI := X[I];
         XMATR[I, 1] := 1.0; (* prima colonna *)
         FOR J := 2 TO NCOL DO (* altre colonne *)
            XMATR[I, J] := XMATR[I,J-1] * XI
      END;
   SQUARE(XMATR, Y, A, G, NROW, NCOL);
   GAUSSJ(A, G, COEF, NCOL, ERROR);
   SUM_Y := 0.0;
   SUM_Y2 := 0.0;
   SRS := 0.0;
```

Figura 7.5 — Introduzione dell'ordine del polinomio dalla console (segue).

```
    FOR I := 1 TO NROW DO
      BEGIN
        YI := Y[I];
        YC := 0.0;
        FOR J := 1 TO NCOL DO
          YC := YC + COEF[J] * XMATR[I, J];
        Y_CALC[I] := YC;
        RESID[I] := YC - YI;
        SRS := SRS + SQR(RESID[I]);
        SUM_Y := SUM_Y + YI;
        SUM_Y2 := SUM_Y2 + YI * YI
      END;
    CORREL_COEF :=
      SQRT(1.0 - SRS/(SUM_Y2 - SQR(SUM_Y)/NROW));
    IF NROW = NCOL THEN NM := 1
    ELSE NM := NROW - NCOL;
    SEE := SQRT(SRS/NM);
    FOR I := 1 TO NCOL DO (* errori nella soluzione *)
      SIG[I] := SEE * SQRT(A[I, I])
  END (* linfit *);
  (* PROCEDURE plot(x, y, z: ary; nrow: integer);
  extern; *)
  (*$F PLOT.PAS *)
  BEGIN (* programma principale *)
    DONE := FALSE;
    WRITELN;
    GET_DATA (X, Y, NROW);
    REPEAT
      REPEAT
        WRITE('Ordine dell'interpolazione polinomiale?');
        READLN( NCOL)
      UNTIL NCOL < 5;
      IF NCOL < 1 THEN DONE := TRUE (* finisce se ncol < 1 *)
      ELSE
```

Figura 7.5 — Introduzione dell'ordine del polinomio dalla console (segue).

```
        BEGIN
          NCOL := NCOL + 1 (* l'ordine è diminuito di uno *)
          LINFIT(X, Y, Y_CALC, RESID, COEF, SIG, NROW, NCOL);
          WRITE_DATA;
          PLOT (X, Y, Y_CALC, NROW)
        END (* ELSE *)
    UNTIL DONE
END.
```

Figura 7.5 — Introduzione dell'ordine del polinomio dalla console (segue).

Confronto tra diverse esecuzioni del programma

Eseguite il programma con il valore 2 per il grado dell'equazione; i risultati saranno ancora gli stessi delle due versioni precedenti, ma questa volta il programma ritorna all'inizio e vi chiede nuovamente il grado dell'equazione. Questa volta rispondete 1; i risultati (mostrati nella Figura 7.6) sono sulla retta che attraversa i punti dell'insieme di dati.

I due coefficienti rappresentano l'equazione:

$$y = 12.028 - 0.240x$$

Notate che circa metà dei punti sono da una parte della retta, e metà dall'altra; la retta passa per i punti come meglio può, e d'altra parte il coefficiente di correlazione è solo 0.096, valore relativamente piccolo che sta a significare che la retta non è una buona interpolazione.

La terza volta date il valore 3; questo produce un'interpolazione corrispondente all'equazione cubica:

$$y = -7.2 + 10.0x - 0.946x^2 - 0.00915x^3$$

Dal valore del coefficiente di correlazione si capisce che la curva risultante non da un'interpolazione migliore di quella parabolica; si dovrebbe sempre usare l'equazione di grado più basso che interpola i dati in modo accettabile, e quindi in questo caso la scelta migliore è la parabola.

Con questa versione del programma possiamo scegliere al massimo il grado 3, in quanto abbiamo fissato a 4 il numero massimo di colonne della matrice, nella costante MAXC; se è necessario un grado maggiore basta cambiare MAXC.

I	X	Y	Y CALC	RESID
1	1.0	2.07	11.79	9.72
2	2.0	8.60	11.55	2.95
3	3.0	14.42	11.31	-3.11
4	4.0	15.80	11.07	-4.73
5	5.0	18.92	10.83	-8.09
6	6.0	17.96	10.59	-7.37
7	7.0	12.98	10.35	-2.63
8	8.0	6.45	10.11	3.66
9	9.0	0.27	9.87	9.60

coefficienti	errori	
1.20275e1	5.26361	Termine costante
-2.39500e-1	9.35368e-1	

Il coefficiente di correlazione è 0.09633

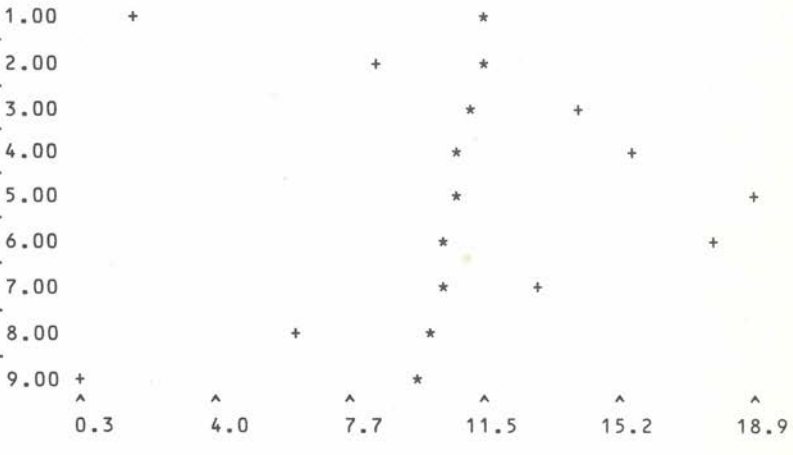


Figura 7.6 — Interpolazione rettilinea per dati parabolici.

Si esce dal programma dando un valore nullo o negativo per il grado. Si potrebbe modificare il programma in modo che chieda il numero dei termini dell'equazione invece che il grado; in questo caso deve essere eliminata dal programma principale l'istruzione:

NCOL : = NCOL + 1; (*l'ordine è inferiore di 1*)

Nei prossimi tre paragrafi studieremo le applicazioni sperimentali del programma di interpolazione al calcolo della capacità termica dell'ossigeno, della pressione del vapore di piombo liquido e delle proprietà del vapore surriscaldato.

PROGRAMMA PASCAL: L'EQUAZIONE DELLA CAPACITÀ TERMICA

La capacità termica è una misura di quanto cresce la temperatura di un corpo quando gli viene fornita una certa quantità di calore; i dati determinati sperimentalmente sono in genere interpolati dall'equazione:

$$C_p = A + BT + \frac{C}{T^2}$$

dove C_p è la capacità termica in unità di energia per grado, e T è la temperatura assoluta; i coefficienti, come al solito, sono A, B e C.

Fate una copia del programma di interpolazione della Figura 7.3 (non della Figura 7.5, che serve in particolare per determinare il grado); modificate la procedura GET_DATA e la LINFIT secondo la versione della Figura 7.7. Come prima, la procedura LINFIT riempie la prima colonna della matrice dei dati con 1, e la seconda con la temperatura (la variabile indipendente); la terza colonna ora contiene l'inverso del quadrato della temperatura.

I dati rappresentano la capacità termica dell'ossigeno in un campo di variazione della temperatura da 300 a 1200 gradi kelvin; notate che nella procedura GET_DATA X e Y sono diventati T e CP, nomi locali alla procedura stessa, che possono quindi essere cambiati quando si vuole.

```
PROCEDURE GET_DATA
          (VAR T     : ARY; (* variabile indipendente *)
           VAR CP    : ARY; (* variabile dipendente *)
           VAR NROW  : INTEGER); (* lunghezza dei vettori *)
VAR
  I: INTEGER;
BEGIN
  NROW := 10;
  FOR I := 1 TO NROW DO
    T[I] := (I + 2) * 100;
  CP[1] := 7.02; CP[2] := 7.20;
  CP[3] := 7.43; CP[4] := 7.67;
```

Figura 7.7 — Procedure GET_DATA e LINFIT per l'equazione della capacità termica

```
    CP[5] := 7.88; CP[6] := 8.06;
    CP[7] := 8.21; CP[8] := 8.34;
    CP[9] := 8.44; CP[10] := 8.53
END (* procedure get_data *);


PROCEDURE LINFIT(X,          (* variabile indipendente *)
                 Y : ARY; (* variabile dipendente *)
      VAR Y_CALC: ARY; (* var. dipen. calcolata *)
      VAR RESID   : ARY; (* matrice dei resti *)
      VAR  COEF   : ARYS; (* coefficienti *)
      VAR  SIG    : ARYS; (* errori nei coefficienti *)
           NROW   : INTEGER; (* lunghezza dei vettori *)
      VAR  NCOL   : INTEGER); (* numero dei termini *)
(* interpolazione coi minimi quadrati di *)
(* nrow coppie di punti x e y *)
(* Sono necessarie le procedure:
   SQUARE — formazione della matrice dei coeffic.
   GAUSSJ — eliminazione di Gauss-Jordan *)

VAR
   XMATR : ARY2; (* matrice dei dati *)
   A     : ARY2S; (* matrice dei coefficienti *)
   G     : ARYS; (* vettore delle costanti *)
   ERROR : BOOLEAN;
   I, J, NM : INTEGER;
   XI, YI, YC, SRS, SEE,
   SUM_Y, SUM_Y2: REAL;

BEGIN (* procedure linfit *)
   NCOL := 3 (* numero dei termini *)
   FOR I := 1 TO NROW DO
```

Figura 7.7 — Procedure GET_DATA e LINFIT per l'equazione della capacità termica (segue).

```
        BEGIN (* caricamento della matrice X *)
          XI := X[I];
          XMATR[I, 1] := 1.0 (* prima colonna *)
          XMATR[I, 2] := XI (* seconda colonna *)
          XMATR[I, 3] := 1.0/SQR(XI) (* terza colonna *)
        END;


    SQUARE(XMATR, Y, A, G, NROW, NCOL);
    GAUSSJ(A, G, COEF, NCOL, ERROR);
    SUM_Y := 0.0;
    SUM_Y2 := 0.0;
    SRS := 0.0;
    FOR I := 1 TO NROW DO
      BEGIN
        YI := Y[I];
        YC := 0.0;
        FOR J := 1 TO NCOL DO
          YC := YC + COEF[J] * XMATR[I, J];
        Y_CALC[I] := YC;
        RESID[I] := YC - YI;
        SRS := SRS + SQR(RESID[I]);
        SUM_Y := SUM_Y + YI;
        SUM_Y2 := SUM_Y2 + YI * YI
      END;
    CORREL_COEF :=
            SQRT(1.0 - SRS/(SUM_Y2 - SQR(SUM_Y)/NROW));
    IF NROW = NCOL THEN NM := 1
    ELSE NM := NROW - NCOL;
    SEE := SQRT(SRS/NM);
    FOR I := 1 TO NCOL DO (* errori nella soluzione *)
      SIG[I] := SEE * SQRT(A[I, I])
  END (* linfit *);
```

*Figura 7.7 — Procedure GET*_DATA e LINFIT per l'equazione della capacità termica (segue).

Compilate il programma ed eseguitelo; i risultati si presenteranno come in Figura 7.8, e l'equazione risultante è:

$$C_p = 6.9 + 0.00143T - \frac{32610}{T^2}$$

PROGRAMMA PASCAL: L'EQUAZIONE DELLA PRESSIONE DI UN VAPORE

Quando un gas o un vapore sono in equilibrio con il liquido e il solido da cui sono prodotti, si dicono saturi; per sostanze pure, la pressione di saturazione è funzione

```
  I        X         Y      Y CALC     RESID
  1    300.0      7.02       6.97     -0.05
  2    400.0      7.20       7.27      0.07
  3    500.0      7.43       7.49      0.06
  4    600.0      7.67       7.67      0.00
  5    700.0      7.88       7.84     -0.04
  6    800.0      8.06       8.00     -0.06
  7    900.0      8.21       8.16     -0.05
  8   1000.0      8.34       8.31     -0.03
  9   1100.0      8.44       8.46      0.02
 10   1200.0      8.53       8.60      0.07

coefficienti        errori
 6.90485e0       1.29216e-1   Termini costanti
 1.43448e-3      1.26803e-4
-3.26106e4       1.16750e4

 Il coefficiente di correlazione è  0.99482

  300.00 *+
   -
  400.00        +  *
   -
  500.00                 **
   -
  600.00                         *
   -
  700.00                            *+
   -
  800.00                                 * +
   -
  900.00                                      *+
   -
 1000.00                                          *+
   -
 1100.00                                              +*
   -
 1200.00                                                 +  *
          ^         ^         ^         ^         ^         ^
         7.0       7.3       7.6       8.0       8.3       8.6
```

Figura 7.8 — Risultati relativi alla capacità termica dell'ossigeno.

della temperatura, e l'equazione che si usa in genere per rappresentare questa relazione è:

$$\log P = A + \frac{B}{T} + C \log T$$

dove P è la pressione e T è la temperatura assoluta, A, B e C sono i coefficienti e log può essere sia il logaritmo naturale che quello comune.

Fate una copia del programma della Figura 7.7, modificate le procedure GET_DATA e LINFIT secondo la versione della Figura 7.9; la procedura LINFIT mette come sempre 1 nella prima colonna della matrice, l'inverso della temperatura (variabile indipendente) nella seconda e il logaritmo della temperatura nella terza.

```
PROCEDURE GET_DATA
          (VAR T    : ARY; (* variabile indipendente *)
           VAR P    : ARY; (* variabile dipendente *)
           VAR NROW : INTEGER); (* lunghezza dei vettori *)

VAR
  I: INTEGER;

BEGIN (* get_data *)
  NROW := 10;
  FOR I := 1 TO NROW DO
    T[I] := (I + 6.0) * 100.0;
  P[1] := 1.0E-9; P[2] := 5.598E-8;
  P[3] := 1.234E-6; P[4] := 1.507E-5;
  P[5] := 1.138E-4; P[6] := 6.067E-4;
  P[7] := 2.512E-3; P[8] := 8.337E-3;
  P[9] := 2.371E-2; P[10] := 5.875E-2;
  FOR I := 1 TO NROW DO
    P[I] := LN(P[I]) (* pone log di p *)
END (* procedure get_data *);
```

Figura 7.9 — Procedure GET_DATA e LINFIT per l'equazione della pressione di un vapore.

```
PROCEDURE LINFIT(X,          (* variabile indipendente *)
                 Y : ARY; (* variabile dipendente *)
    VAR Y_CALC: ARY; (* var. dipendente calcolata *)
    VAR RESID   : ARY; (* matrice dei resti *)
    VAR  COEF   : ARYS; (* coefficienti *)
    VAR  SIG    : ARYS; (* errori nei coefficienti *)
         NROW   : INTEGER; (* lunghezza delle matrici *)
    VAR  NCOL   : INTEGER); (* numeri di termini *)

(* interpolazione coi minimi quadrati di *)
(* nrow coppie di punti x e y *)
(* Sono richieste le procedure separate:
   SQUARE — formazione della matrice dei coefficienti
   GAUSSJ — eliminazione di Gauss-Jordan *)


VAR
  XMATR  : ARY2; (* matrice dei dati *)
  A      : ARY2S; (* matrice dei coefficienti *)
  G      : ARYS; (* vettore delle costanti *)
  ERROR  : BOOLEAN;
  I, J, NM : INTEGER;
  XI, YI, YC, SRS, SEE,
  SUM_Y, SUM_Y2: REAL;

BEGIN (* procedure linfit *)
  NCOL := 3 (* numero di termini *)
  FOR I := 1 TO NROW DO
    BEGIN (* carica la matrice X *)
      XI := X[I];
      XMATR[I, 1] := 1.0 (* prima colonna *)
```

Figura 7.9 — Procedure GET_DATA e LINFIT per l'equazione della pressione di un vapore (segue).

```
        XMATR[I, 2] := 1.0/XI (* seconda colonna *)
        XMATR[I, 3] := LN(XI) (* terza colonna *)
      END;
    SQUARE(XMATR, Y, A, G, NROW, NCOL);
    GAUSSJ(A, G, COEF, NCOL, ERROR);
    SUM_Y := 0.0;
    SUM_Y2 := 0.0;
    SRS := 0.0;
    FOR I := 1 TO NROW DO
      BEGIN
        YI := Y[I];
        YC := 0.0;


        FOR J := 1 TO NCOL DO
          YC := YC + COEF[J] * XMATR[I, J];
        Y_CALC[I] := YC;
        RESID[I] := YC - YI;
        SRS := SRS + SQR(RESID[I]);
        SUM_Y := SUM_Y + YI;
        SUM_Y2 := SUM_Y2 + YI * YI
      END;
    CORREL_COEF :=
      SQRT(1.0 - SRS/(SUM_Y2 - SQR(SUM_Y)/NROW));
    IF NROW = NCOL THEN NM := 1
    ELSE NM := NROW - NCOL;
    SEE := SQRT(SRS/NM);
    FOR I := 1 TO NCOL DO (* errori nella soluzione *)
      SIG[I] := SEE * SQRT(A[I, I])
  END (* linfit *);
```

Figura 7.9 — Procedure GET_DATA e LINFIT per l'equazione della pressione di un vapore (segue).

Eseguite il programma; i risultati si presenteranno come in Figura 7.10.

```
 I        X         Y       Y CALC     RESID
 1    700.0    -20.72    -20.72     -0.00
 2    800.0    -16.70    -16.70     -0.01
 3    900.0    -13.61    -13.59      0.02
 4   1000.0    -11.10    -11.11     -0.01
 5   1100.0     -9.08     -9.09     -0.00
 6   1200.0     -7.41     -7.41      0.00
 7   1300.0     -5.99     -5.99     -0.01
 8   1400.0     -4.79     -4.78      0.00
 9   1500.0     -3.74     -3.74      0.00
10   1600.0     -2.83     -2.83      0.00

coefficienti          errori
 1.81432e1          5.96613e-1   Termine costante
-2.31700e4          7.77388e1
-8.80333e-1         7.48010e-2

 Il coefficiente di correlazione è  1.00000
```

Figura 7.10 — Risultati relativi alla pressione del vapore del piombo.

I dati rappresentano la pressione del vapore del piombo liquido, al variare della temperatura da 700 a 1600 °K; l'equazione corrispondente è:

$$\ln P = 18.17 - \frac{23174}{T} - 0.8803 \ln T$$

dove la pressione è misurata in atmosfere e ln è il logaritmo naturale; il coefficiente di correlazione 1.00000 significa che l'interpolazione è molto buona.

Notate che le pressioni originali nella procedura GET_DATA sono in atmosfere; nella procedura stessa sono poi convertite nei loro logaritmi. Lasciamo come esercizio al lettore di conservare i valori originali, usando, ad esempio, un'altra matrice di nome PRESS; la procedura WRITE_DATA può allora essere modificata in modo che stampi sia i valori delle pressioni che i loro logaritmi.

Nell'ultimo esempio tratteremo un'equazione che non rispetta le condizioni stabilite all'inizio del capitolo; uno dei coefficienti delle incognite non è lineare, e precisamente è un esponenziale. Più avanti, nel Capitolo 10, studieremo l'algoritmo per trattare questa equazione, ma qui ci accontenteremo di una soluzione meno elegante, e cioè proveremo con diversi valori dell'esponente, fin quando troveremo il migliore coefficiente di correlazione.

UNA EQUAZIONE A TRE VARIABILI

Nel paragrafo precedente abbiamo studiato l'equazione di stato di un gas saturo, per cui la pressione è funzione della temperatura. Consideriamo ora l'equazione di stato per un gas surriscaldato; la temperatura in questo caso è maggiore di quella di saturazione, o, da un altro punto di vista, la pressione è inferiore a quella di saturazione. Poichè in queste condizioni la temperatura e la pressione sono variabili indipendenti, possiamo esprimere il volume in funzione di entrambe.

Per un gas ideale l'equazione di stato è:

$$VP = RT$$

dove V è il volume molare, T è la temperatura, P è la pressione e R la costante dei gas. Ma all'aumentare della pressione, il comportamento del gas si allontana sempre più da quello di un gas ideale, e l'equazione diventa:

$$PV = A + BP + CP^2 + DP^3 + \ldots$$

che non è altro che lo sviluppo in serie di potenze della pressione.

Fate attenzione a non confondere la costante del gas (R) con i residui (r); nel programma la costante dei gas si chiama R e i residui RESID.

Poichè tutti i gas diventano ideali quando la pressione diminuisce sotto un certo valore, l'equazione di stato deve gradualmente trasformarsi in quella dei gas ideali, man mano che la pressione si avvicina allo zero; quindi il valore di A nell'equazione precedente deve essere uguale a RT. Poichè a basse pressioni basta un minor numero di termini polinomiali, l'equazione di stato viene spesso scritta così:

$$PV = RT + BP + CP^2$$

Il calcolo dei coefficienti B e C è immediato quando la temperatura è costante; se invece è variabile, B e C sono funzioni della temperatura.

Diverse equazioni di stato vengono usate comunemente, tutte per altro empiriche; come esempio di equazione a tre variabili, consideriamo l'espressione:

$$PV = RT + \frac{BP}{T^n} + CP^2$$

In questo caso C non è funzione di T, ma il coefficiente originale B è diventato la funzione:

$$\frac{B}{T^n}$$

Questa equazione ha un coefficiente non lineare n, e non può quindi essere risolta con i metodi visti finora; se però assegnamo un valore ipotetico a n, possiamo deter-

minare gli altri coefficienti lineari. Cominceremo con l'assegnare il valore 1 a n, e calcoleremo gli altri coefficienti; a questo punto potremo valutare con che fedeltà l'equazione risultante rappresenta i dati originali. Cambiando i valori di n vedremo quale equazione è la migliore.

PROGRAMMA PASCAL: UN'EQUAZIONE DI STATO PER IL VAPORE

Il programma di Figura 7.11 può essere usato per trovare i coefficienti B e C; poichè il coefficiente del primo termine a destra è 1, l'equazione può essere riscritta:

$$PV - RT = \frac{BP}{T^n} + CP^2$$

I dati sono definiti nella GET_DATA; la temperatura è data in grandi Fahrenheit, la pressione in libbre per pollici quadrati e il volume specifico in piedi cubici per libbre massa. Le temperature sono convertite nella scala Rankine assoluta sommando 460, mentre le pressioni restano in libbre per pollici quadrati; la costante R, che per il vapore vale 85.76, viene moltiplicata per 144 pollici quadrati per piedi quadrati.

La matrice dei dati è creata nella procedura LINFIT, come al solito; la prima colonna contiene la pressione divisa per la potenza n-esima della temperatura, la seconda contiene il quadrato della pressione; il vettore y contiene i valori PV - RT.

Notate che è la prima volta che non abbiamo 1 nella prima colonna della matrice. Potremmo naturalmente dividere l'equazione per la pressione, e il secondo membro diventerebbe di primo grado; ma avremmo dei problemi quando la pressione si avvicina a zero.

Confronto di diverse esecuzioni per determinare il valore di n

Scrivete il programma della Figura 7.11 ed eseguitelo; notate che l'esponente n è inizialmente a 1, e i risultati si presentano come in Figura 7.12.

Dalla figura 7.12 si vede che i risultati non sono molto buoni. Il coefficiente di correlazione è 92%, ma alcuni dei valori calcolati si allontanano di più del 10% da quelli originali. Tenteremo un'altra soluzione; modifichiamo la variabile POWER all'inizio del programma per dare il valore 2 all'esponente n, e precisamente poniamo POWER uguale al quadrato della temperatura.

```
POWER : = SQR ( T[I]);
```

```
PROGRAM LEAST6(OUTPUT);
(* 8 Feb 81 *)
(* Programma pascal che esegue una *)
(* interpolazione coi minimi quadrati *)
(* sulle proprietà del valore *)
(* con la routine Gauss-Jordan *)
(* Sono necessarie le procedure separate:
          GAUSSJ, PLOT *)

CONST
  MAXR = 20 (* punti *);
  MAXC = 4 (* termini polinomiali *);
TYPE
  ARY   = ARRAY[1..MAXR] OF REAL;
  ARYS  = ARRAY[1..MAXC] OF REAL;
  ARY2  = ARRAY[1..MAXR, 1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXC, 1..MAXC] OF REAL;
VAR
  P, T, V, Y,
  Y_CALC, RESID : ARY;
  COEF, SIG     : ARYS;
  NROW, NCOL    : INTEGER;
  CORREL_COEF   : REAL;
PROCEDURE GET_DATA
          (VAR P, T    : ARY; (* variabile indipendente *)
          VAR V        : ARY; (* variabile dipendente *)
          VAR NROW     : INTEGER); (* lunghezza dei vettori *)
VAR
  I: INTEGER;
BEGIN
  NROW := 12;
  T[1] := 400; P[1] := 120; V[1] := 4.079;
  T[2] := 450; P[2] := 120; V[2] := 4.36;
  T[3] := 500; P[3] := 120; V[3] := 4.633;
```

Figura 7.11 — Un'equazione di stato per il vapore.

```
  T[4] := 400; P[4] := 140; V[4] := 3.466;
  T[5] := 450; P[5] := 140; V[5] := 3.713;
  T[6] := 500; P[6] := 140; V[6] := 3.952;
  T[7] := 400; P[7] := 160; V[7] := 3.007;
  T[8] := 450; P[8] := 160; V[8] := 3.228;
  T[9] := 500; P[9] := 160; V[9] := 3.44;
  T[10] := 400; P[10] := 180; V[10] := 2.648;
  T[11] := 450; P[11] := 180; V[11] := 2.85;
  T[12] := 500; P[12] := 180; V[12] := 3.042;
  FOR I := 1 TO NROW DO
    T[I] := T[I] + 460.0 (* conversione a Rankine *)
END (* procedure get_data *);

PROCEDURE WRITE_DATA;
(* stampa i risultati *)
VAR
  I: INTEGER;
BEGIN
  WRITELN;
  WRITELN(' I  P   T   V   ',
              'Y   Y CALC   %RES');
  FOR I := 1 TO NROW DO
    WRITELN(I:3, P[I]:7:1, T[I]:7:1, V[I]:7:3,
    Y[I]:9:2,Y_CALC[I]:9:2, (100.0 * RESID[I]/Y[I]):9:2);
  WRITELN;
  WRITELN('Coefficienti errori');
  WRITELN(COEF[1], ' ', SIG[1], 'Termine costante');
  FOR I := 2 TO NCOL DO
    WRITELN(COEF[I], ' ', SIG[I]) (* altri termini *);
  WRITELN;
  WRITELN
    ('Il coefficiente di correlazione è', CORREL_COEF: 8: 5)
END (* write_data *);
```

Figura 7.11 — Un'equazione di stato per il vapore (segue).

```
(* PROCEDURE square(x : ary2;
                     y : ary;
                 VAR a : ary2s;
                 VAR g : arys;
            nrow,ncol : integer);
extern; *)
(*$F SQUARE.PAS *)

(* PROCEDURE gaussj
   (VAR b     : ary2s;
        y     : arys;
    VAR coef  : arys;
        ncol  : integer;
    VAR error : boolean);
extern; *)
(*$F GAUSSJ.PAS *)

PROCEDURE LINFIT
          (P, T, V  :        (* variabile indipendente *)
      VAR Y         : ARY; (* variabile dipendente *)
      VAR Y_CALC: ARY; (* var. dipendente calcolata *)
      VAR RESID     : ARY; (* matrice dei resti *)
      VAR  COEF     : ARYS; (* coefficienti *)
      VAR  SIG      : ARYS; (* errori nei coefficienti *)
           NROW     : INTEGER; (* lunghezza delle matrici *)
      VAR  NCOL     : INTEGER); (* numero di termini *)
(* interpola una equazione di stato attraverso
   nrow gruppi di p, t e v gruppi di punti *)
(* Sono necessarie le procedure separate:
   SQUARE — formazione matrice dei coefficienti
   GAUSSJ — eliminazione di Gauss-Jordan *)
CONST
   R = 85.76 (* costante del gas per il vapore *)
```

Figura 7.11 — Un'equazione di stato per il vapore (segue).

```
VAR
   XMATR  : ARY2; (* matrice dei dati *)
   A      : ARY2S; (* matrice dei coefficienti *)
   G      : ARYS; (* vettore delle costanti *)
   ERROR  : BOOLEAN;
   I, J, NM : INTEGER;
   POWER, YI, YC, SRS,
   SEE, SUM_Y, SUM_Y2: REAL;
BEGIN (* procedure linfit *)
   NCOL := 2 (* numero di termini *);
   FOR I := 1 TO NROW DO
      BEGIN (* carica la matrice X *)
         POWER := T[I];
         XMATR[I, 1] := P[I]/POWER (* prima colonna *);
         XMATR[I, 2] := SQRT(P[I]) (* seconda colonna *);
         Y[I] := V[I] * P[I] - R * T[I]/144.0
      END;
   SQUARE(XMATR, Y, A, G, NROW, NCOL);
   GAUSSJ(A, G, COEF, NCOL, ERROR);
   SUM_Y := 0.0;
   SUM_Y2 := 0.0;
   SRS := 0.0;
   FOR I := 1 TO NROW DO
      BEGIN
         YI := Y[I];
         YC := 0.0;
         FOR J := 1 TO NCOL DO
            YC := YC + COEF[J] * XMATR[I, J];
         Y_CALC[I] := YC;
         RESID[I] := YC - YI;
         SRS := SRS + SQR(RESID[I]);
         SUM_Y := SUM_Y + YI;
         SUM_Y2 := SUM_Y2 + YI * YI
      END;
```

Figura 7.11 — Un'equazione di stato per il vapore (segue).

```
    CORREL_COEF :=
        SQRT(1.0 - SRS/(SUM_Y2 - SQR(SUM_Y)/NROW));
    IF NROW = NCOL THEN NM := 1
    ELSE NM := NROW - NCOL;
    SEE := SQRT(SRS/NM);
    FOR I := 1 TO NCOL DO (* errori nella soluzione *)
        SIG[I] := SEE * SQRT(A[I, I])
END (* linfit *);
BEGIN (* programma principale *)
    GET_DATA(P, T, V, NROW);
    LINFIT(P, T, V, Y, Y_CALC, RESID, COEF, SIG, NROW, NCOL);
    WRITE_DATA
END.
```

Figura 7.11 — Un'equazione di stato per il vapore (segue).

I	P	T	V	Y	Y CALC	%RES
1	120.0	860.0	4.079	-22.70	-19.44	-14.36
2	120.0	910.0	4.360	-18.76	-17.43	-7.07
3	120.0	960.0	4.633	-15.77	-15.63	-0.92
4	140.0	860.0	3.466	-26.94	-24.16	-10.30
5	140.0	910.0	3.713	-22.14	-21.82	-1.44
6	140.0	960.0	3.952	-18.45	-19.72	6.85
7	160.0	860.0	3.007	-31.06	-28.98	-6.69
8	160.0	910.0	3.228	-25.48	-26.30	3.24
9	160.0	960.0	3.440	-21.33	-23.90	12.03
10	180.0	860.0	2.648	-35.54	-33.88	-4.67
11	180.0	910.0	2.850	-28.96	-30.86	6.59
12	180.0	960.0	3.042	-24.17	-28.16	16.50

coefficienti	errori	
-2.62187e2	4.75869e1	Termine costante
1.56514e0	6.49499e-1	

Il coefficiente di correlazione è 0.91839

Figura 7.12 — Proprietà del vapore surriscaldato (n = 1).

Eseguite nuovamente il programma; i risultati si presenteranno come in Figura 7.13.

La curva risultante è meglio di prima; il coefficiente di correlazione è 98%, e i valori calcolati sono tutti in un intervallo del 5% intorno a quelli originali. Possiamo continuare così aumentando a 3 l'esponente, sostituendo alla precedente:

```
POWER : = T[I]*SQR(T[I]);
```

Eseguite il programma una terza volta, e confrontate i risultati con la Figura 7.14.

I	P	T	V	Y	Y CALC	%RES
1	120.0	860.0	4.079	-22.70	-21.49	-5.32
2	120.0	910.0	4.360	-18.76	-18.03	-3.85
3	120.0	960.0	4.633	-15.77	-15.10	-4.25
4	140.0	860.0	3.466	-26.94	-26.01	-3.44
5	140.0	910.0	3.713	-22.14	-21.98	-0.71
6	140.0	960.0	3.952	-18.45	-18.56	0.58
7	160.0	860.0	3.007	-31.06	-30.59	-1.49
8	160.0	910.0	3.228	-25.48	-25.98	2.00
9	160.0	960.0	3.440	-21.33	-22.08	3.48
10	180.0	860.0	2.648	-35.54	-35.22	-0.88
11	180.0	910.0	2.850	-28.96	-30.04	3.74
12	180.0	960.0	3.042	-24.17	-25.64	6.08

coefficienti	errori	
-1.99359e5	1.19367e4	Termine costante
9.90932e-1	1.80298e-1	

Il coefficiente di correlazione è 0.98901

Figura 7.13 — Proprietà del vapore surriscaldato (n = 2).

I	P	T	V	Y	Y CALC	%RES
1	120.0	860.0	4.079	-22.70	-22.88	0.78
2	120.0	910.0	4.360	-18.76	-18.80	0.22
3	120.0	960.0	4.633	-15.77	-15.52	-1.59
4	140.0	860.0	3.466	-26.94	-26.97	0.13
5	140.0	910.0	3.713	-22.14	-22.21	0.35
6	140.0	960.0	3.952	-18.45	-18.39	-0.32
7	160.0	860.0	3.007	-31.06	-31.09	0.10
8	160.0	910.0	3.228	-25.48	-25.65	0.68
9	160.0	960.0	3.440	-21.33	-21.28	-0.23
10	180.0	860.0	2.648	-35.54	-35.22	-0.90
11	180.0	910.0	2.850	-28.96	-29.10	0.49
12	180.0	960.0	3.042	-24.17	-24.19	0.06

coefficienti	errori	
-1.38664e8	1.50165e6	Termine costante
2.99897e-1	2.52008e-2	

Il coefficiente di correlazione è 0.99963

Figura 7.14 — Proprietà del vapore surriscaldato (n = 3).

Il risultato è indiscutibilmente migliore; i valori calcolati sono in un intervallo dell'1% intorno a quelli originali, e il coefficiente di correlazione è 99.96%. È il caso di tenere per buoni questi risultati, ma per toglierci ogni dubbio, portiamo a 4 l'esponente, con l'istruzione:

```
POWER : = SQR(SQR(T[I]) );
```

Eseguite il programma e confrontate i risultati con la Figura 7.15; siamo andati troppo lontano. Infatti i punti calcolati sono più lontani di prima da quelli originali e il coefficiente di correlazione è più lontano dall'unità; la procedura di Gauss-Jordan potrebbe dirci che la matrice è singolare, e il problema sta nel fatto che è mal condizionata, come si vede confrontando i quadrati dei due errori standard, il cui rapporto è di circa venti ordini di grandezza.

Potremmo provare a migliorare la situazione con esponenti non interi, prossimi a 3, ma i risultati ci mostrerebbero che 3 è il valore migliore. Un'altra possibilità è di rendere il coefficiente C funzione della temperatura, ma poichè il risultato con 3 è sufficientemente buono, possiamo accontentarci.

I	P	T	V	Y	Y CALC	%RES
1	120.0	860.0	4.079	-22.70	-23.69	4.38
2	120.0	910.0	4.360	-18.76	-19.32	3.02
3	120.0	960.0	4.633	-15.77	-16.00	1.45
4	140.0	860.0	3.466	-26.94	-27.46	1.94
5	140.0	910.0	3.713	-22.14	-22.36	1.02
6	140.0	960.0	3.952	-18.45	-18.49	0.19
7	160.0	860.0	3.007	-31.06	-31.22	0.51
8	160.0	910.0	3.228	-25.48	-25.39	-0.34
9	160.0	960.0	3.440	-21.33	-20.96	-1.74
10	180.0	860.0	2.648	-35.54	-34.96	-1.62
11	180.0	910.0	2.850	-28.96	-28.41	-1.89
12	180.0	960.0	3.042	-24.17	-23.43	-3.08

coefficienti	errori	
-9.84751e10	3.64805e9	Termine costante
-1.90703e-1	6.82835e-2	

Il coefficiente di correlazione è 0.99571

Figura 7.15 — Proprietà del vapore surriscaldato (n = 4).

SOMMARIO

In questo capitolo abbiamo presentato il concetto di generalizzazione dei programmi. Abbiamo visto diverse versioni del programma di interpolazione:

- interpolazione lineare
- interpolazione parabolica
- interpolazione parabolica con approccio per matrici
- interpolazione polinomiale generale con grado variabile del polinomio
- interpolazione con equazione non polinomiale

Tutti questi casi richiedono come unica condizione che i coefficienti siano lineari; nell'ultimo esempio del capitolo, invece, abbiamo visto come eliminare anche questa condizione.

Dobbiamo comunque sviluppare altri metodi per trattare equazioni con coefficienti non lineari.

CAPITOLO 8

RISOLUZIONE DI EQUAZIONI CON IL METODO DI NEWTON

INTRODUZIONE

In questo capitolo svilupperemo un programma per risolvere le equazioni con una tecnica particolare, che prende il nome di metodo di Newton o di Newton Raphson; è un metodo che si presta soprattutto a trovare radici particolari di funzioni continue e limitate. Useremo il metodo di Newton nel Capitolo 10, quando riparleremo dell'interpolazione con equazioni e coefficienti non lineari; naturalmente il metodo di Newton ha molte altre applicazioni, e converrebbe quindi studiarlo anche se non dovessimo usarlo nel resto di questo libro.

Studieremo prima la formulazione matematica di questo metodo, e poi una serie di programmi Pascal, via via più complessi, per implementarlo sul calcolatore. Studieremo in particolare due casi critici di questo metodo, il caso in cui la tangente alla curva ha pendenza zero, e quello in cui successive approssimazioni non convergono a una radice, e vedremo come trattare questi due casi col nostro programma. Incontreremo anche la funzione SIN del Pascal, di cui ci siamo occupati già nel Capitolo 1. Infine useremo il nostro programma per una applicazione pratica, l'equazione della pressione di un vapore.

FORMULAZIONE MATEMATICA DEL METODO DI NEWTON

Cominciamo con il considerare un'equazione del tipo:

$$f(x) = 0 \tag{1}$$

Può avere una soluzione, diverse soluzioni o nessuna. In pratica possono esserci

dei valori di x per cui la f(x) diventa uguale a zero; questi valori prendono il nome di radici o soluzioni dell'equazione, e per valori diversi f(x) sarà diversa da zero.

In certi casi l'equazione si può risolvere esplicitamente. Ad esempio l'espressione:

$$x^2 - 4 = 0$$

può essere convertita in:

$$x^2 = 4$$

che ha le soluzioni:

$$x = 2 \text{ e } x = -2$$

Ma qualche volta l'equazione non può essere risolta così facilmente, come nel caso della seguente espressione:

$$\ln P = A + \frac{B}{T} + C \ln T$$

Questa formula, che trasformeremo in un programma alla fine del capitolo, descrive la pressione del vapore di una sostanza; P è la pressione, T è la temperatura, A, B e C sono costanti diverse per ogni sostanza. Per il piombo, i coefficienti determinati sperimentalmente sono:

$$A = 18.19$$
$$B = -23180$$
$$C = -0.8858$$

quando la pressione è data in atmosfere e la temperatura in gradi Kelvin.

Possiamo facilmente trovare la pressione dei vapori di piombo a 1000 °K risolvendo l'equazione:

$$\ln P = A + \frac{B}{1000} + C \ln 1000$$

Ma supponiamo di voler trovare la temperatura che corrisponde a una pressione di 0.1 atmosfere; dobbiamo risolvere l'equazione:

$$\ln 0.1 = 18.19 - \frac{23180}{T} - 0.8858 \ln T$$

che non è lineare, e non può essere risolta esplicitamente; possiamo però usare un metodo di approssimazione per calcolare la soluzione con quanta precisione vogliamo.

In generale noi scriviamo:

$$y = f(x)$$

e cerchiamo i valori di x per cui y diventa uguale a zero, cioè i punti in cui la curva incontra l'asse x.

Consideriamo ad esempio la curva di equazione:

$$y = f(x) = x^2 - 4$$

che incontra l'asse x in due punti, +2 e −2 come si vede nella Figura 8.1; oppure la curva:

$$y = f(x) = x^2$$

Questa curva è tangente all'asse x nell'origine, dove ha una radice, $x = 0$, come si vede dalla Figura 8.2.

Infine consideriamo l'equazione:

$$y = f(x) = x^2 + 4$$

della Figura 8.3; la curva non incontra l'asse x, e non ha quindi radici reali.

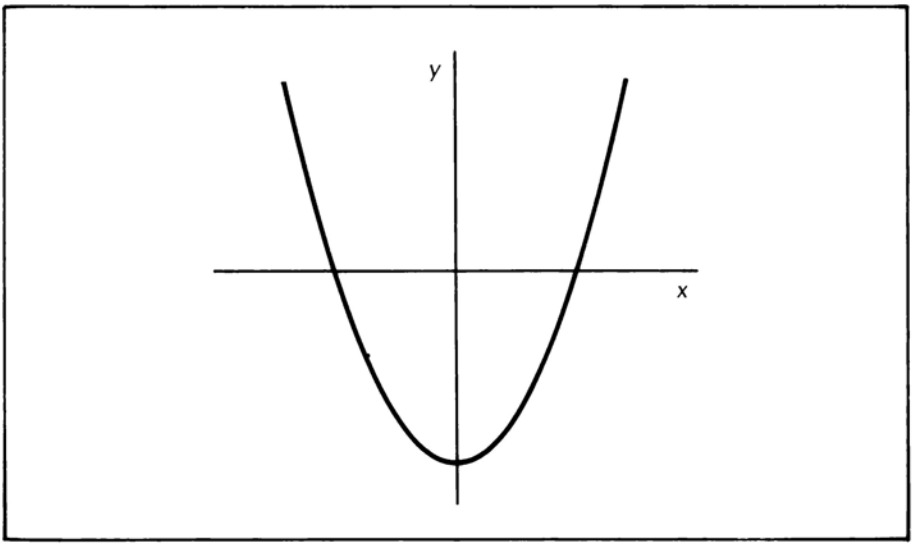


Figura 8.1 — Una funzione con due soluzioni.

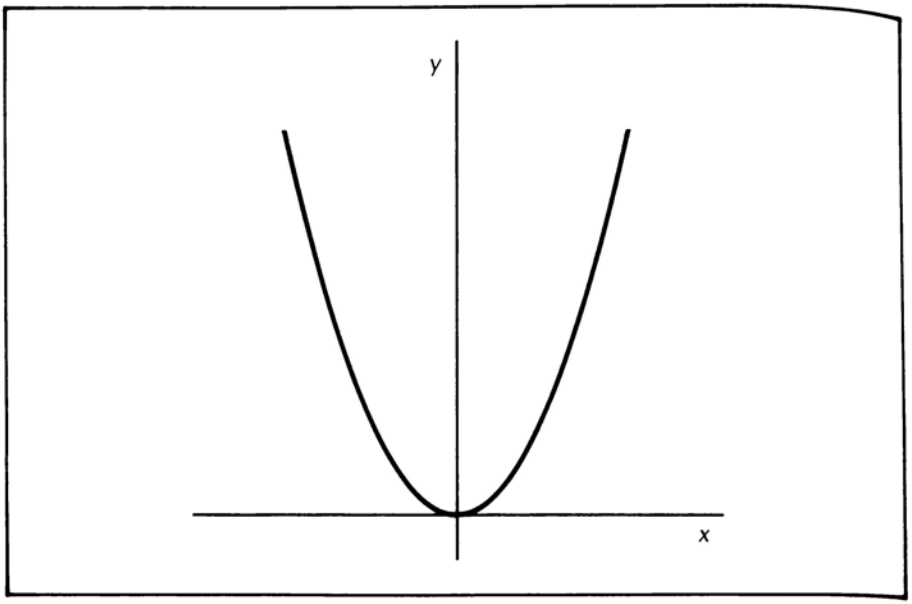


Figura 8.2 — Una funzione con una soluzione.

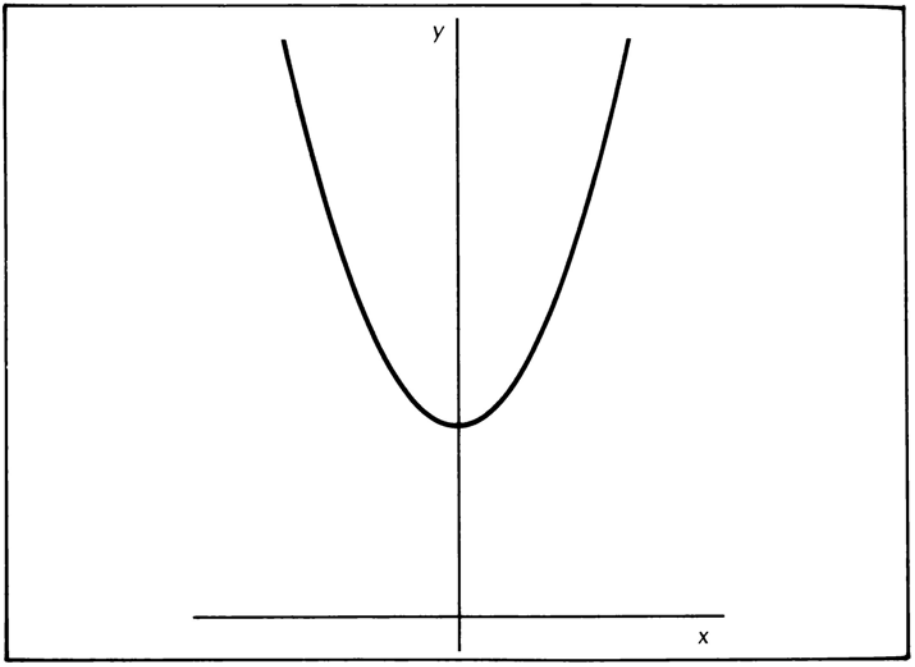


Figura 8.3 — Una funzione senza radici reali.

Studiamo il comportamento di una funzione

$$y = f(x)$$

in un intorno di una radice; potremmo trovare un andamento analogo a quello di Figura 8.4.

La curva incontra l'asse x in corrispondenza della radice, in quanto la relazione:

$$y = f(x) = 0$$

è soddisfatta in quel punto.

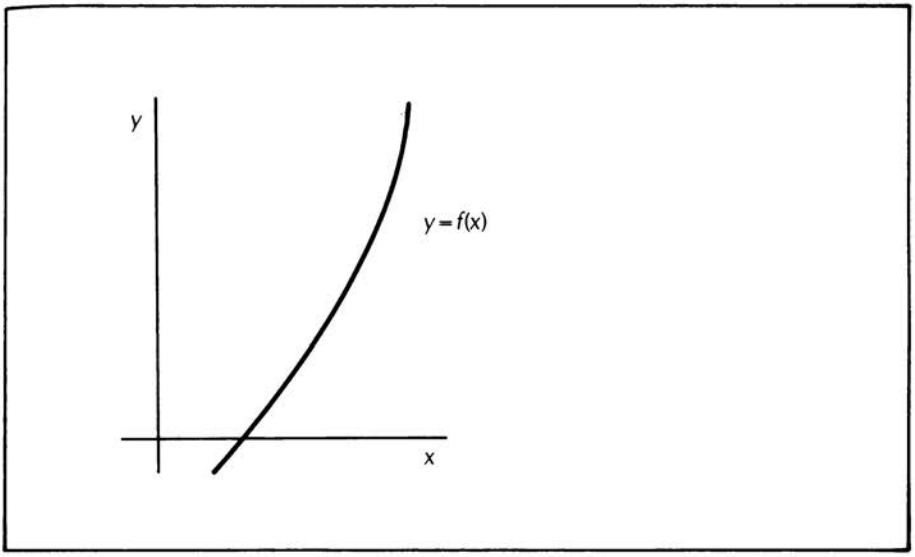


Figura 8.4 — $f(x) = 0$ dove la curva incontra l'asse x.

Una serie di tangenti alla curva $y = f(x)$

Il metodo di Newton comincia col prendere un valore di x, x_1, prossimo a una radice; possiamo determinare il corrispondente valore di y dall'equazione $y_1 = f(x_1)$. La coppia x_1, y_1 rappresenta un punto sulla curva che non è in genere una radice. Si costruisce poi la tangente a f(x) in questo punto, la si prolunga fino a incontrare l'asse x e si prende il valore di questa intersezione (x_2) come seconda approssimazione alla radice; nella Figura 8.5 si vede che x_2 è più vicino alla radice di x_1, e rappresenta quindi un'approssimazione migliore della prima.

Il procedimento viene ripetuto: si calcola il valore della funzione per $x = x_2$ per ot-

tenere il corrispondente valore di y, $y_2 = f(x_2)$; y_2 è minore di y_1, il che significa che è più vicino alla radice. Viene costruita la tangente nel punto $(x_2, f(x_2))$, e la sua intersezione con l'asse x è x_3, terza approssimazione. Possiamo continuare così fino a trovare un valore di x che approssima la radice quanto vogliamo.

Facciamo un passo indietro, e analizziamo nel dettaglio il primo passo. La prima approssimazione x_1 corrisponde a $y_1 = f(x_1)$; la tangente costruita in questo punto ha una pendenza:

$$f'(x_1) = \frac{y_1}{x_1 - x_2} \tag{2}$$

poichè $y_1 = f(x_1)$, l'equazione 2 si può esprimere come:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \tag{3}$$

o, più in generale, come:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{4}$$

dove x_i è la i-esima approssimazione. L'equazione 4 è la formula generale che descrive il metodo di Newton, e può essere una tecnica ideale per trovare la radice di un'equazione reale.

Questo metodo presenta alcuni problemi, che considereremo più avanti; comunque le equazioni che descrivono l'andamento di fenomeni reali hanno tipicamente una sola radice significativa, mentre le altre radici sono negative, nulle o complesse.

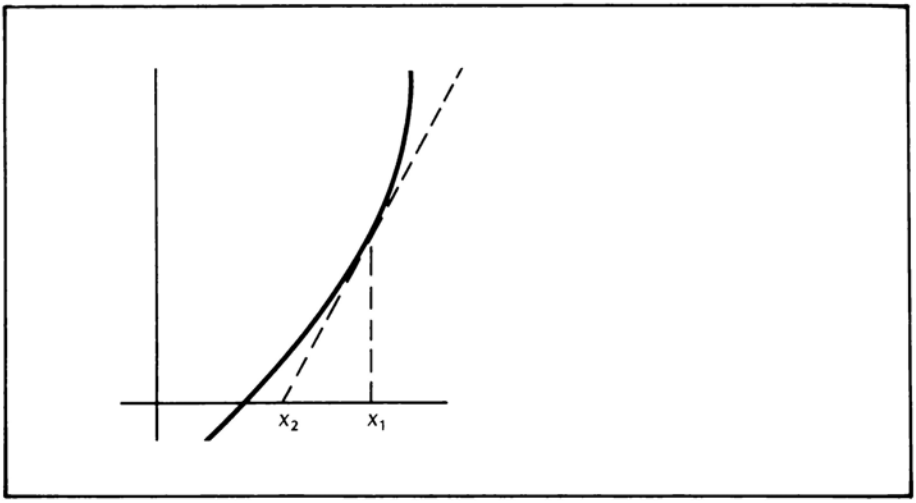


Figura 8.5 — La tangente incrocia l'asse delle x più vicino alla radice della prima approssimazione per x.

Inoltre può darsi che il valore approssimato della risposta sia già noto; ad esempio la legge dei gas ideali fornisce una prima approssimazione di una più complessa equazione di stato.

Avendo determinato un'equazione generale che descrive il metodo di Newton, sarà relativamente facile sviluppare il nostro programma.

PROGRAMMA PASCAL: PRIMA APPLICAZIONE DEL METODO DI NEWTON

Applicheremo il metodo di Newton ad un caso semplice, di cui conosciamo già la risposta. L'equazione da risolvere è:

$$x^2 = 2$$

o

$$x^2 - 2 = 0 \tag{5}$$

la cui soluzione positiva è, naturalmente, la radice quadrata di 2.

Definiamo prima di tutto la funzione:

$$y = f(x) = x^2 - 2 \tag{6}$$

e la sua derivata:

$$\frac{dy}{dx} = f'(x) = 2 \tag{7}$$

In Figura 8.6 c'è il programma che risolve questa equazione con il metodo di Newton. L'algoritmo è contenuto nella procedura Newton, mentre una procedura a parte, la FUNC, esegue il calcolo della funzione (6) e della sua derivata (7). Il corpo del programma principale fornisce la prima approssimazione; poi chiama la procedura Newton per trovare la soluzione e stampare i risultati.

Una buona regola di programmazione vuole che le procedure e le funzioni usate per il calcolo di certi valori non stampino risultati intermedi; ma in questo caso è istruttivo osservare i successivi valori di x, f(x) e f'(x) durante il procedimento, perciò abbiamo messo un'istruzione di stampa nella procedura Newton; quando il programma è stato provato potete eliminarla.

```
PROGRAM NEWDR(OUTPUT);
(* Versione 1, 23 Gen 81 *)
VAR
   X, X2        : REAL;
   ALLDONE      : BOOLEAN;
   ERROR        : BOOLEAN;

PROCEDURE FUNC(X  : REAL;
          VAR FX, DFX : REAL);

BEGIN
   FX  := X * X - 2.0;
   DFX := 2.0 * X
END (* func *);

PROCEDURE NEWTON
            (VAR X: REAL);
CONST
   TOL = 1.0E-6;

VAR
   FX, DFX, DX, X1 : REAL;

BEGIN (* newton *)
   REPEAT
      X1 := X;
      FUNC(X, FX, DFX);
      DX := FX/DFX;
      X := X1 -,DX;
      WRITELN
      ('x= ', X1, ', fx= ', FX, ', dfx= ', DFX)
   UNTIL ABS(DX) <= ABS(TOL * X)
END (* newton *);
```

Figura 8.6 — Metodo di Newton: prima versione

```
BEGIN (* programma principale *)
   WRITELN;
   X := 2.0 (* prima supposizione *);
   NEWTON(X);
   WRITELN;
   WRITELN ('La soluzione è', X);
   WRITELN
END.
```

Figura 8.6 — Metodo di Newton: prima versione (segue).

Tolleranza

Restano da fare alcune considerazioni: le approssimazioni successive sono realizzate nella procedura NEWTON con un ciclo REPEAT/UNTIL, che viene ripetuto fin quando si trovano due valori successivi nella tolleranza desiderata. Non ci interessa il segno della differenza (DX) tra due successive approssimazioni, ma solo la sua ampiezza, che dobbiamo quindi conservare.

Inoltre non ci interessa la differenza effettiva, ma solo quella relativa; se, per esempio, vogliamo una risposta precisa a un milionesimo (1 su 10^6), e una soluzione dell'equazione è 1, allora due valori successivi non dovranno differire per più di 10^{-6}, mentre se la soluzione è 10^{-6} non dovranno differire per più di 10^{-12}. Abbiamo scelto quindi un criterio relativo, invece che assoluto, per porre fine al procedimento iterativo.

Generalizzazione delle chiamate alle procedure

Dobbiamo fare anche alcune considerazioni sulla relazione tra la procedura NEWTON e la procedura da lei richiamata, FUNC, che calcola la funzione e le sue derivate. La NEWTON dà le direttive per eseguire le operazioni descritte dall'equazione 4, in modo indipendente dalla funzione su cui sta operando. Perciò, la FUNC deve essere una entità completamente separata.

In teoria, il nome della procedura FUNC nella NEWTON, dovrebbe essere un parametro dummy, mentre il nome vero dovrebbe essere passato alla NEWTON come

parametro effettivo durante la sua esecuzione. In questo modo, la NEWTON potrebbe risolvere direttamente diverse equazioni. Questo è facilmente realizzabile in FORTAN, mentre i comuni compilatori Pascal non consentono in genere di passare nomi di procedure come parametri ad altre procedure.

Se due o più diverse equazioni devono essere risolte con lo stesso programma, dobbiamo fornire diverse copie separate della procedura NEWTON, ognuna naturalmente con un nome diverso. Una tecnica più complicata sarebbe quella di usare un'istruzione CASE nella FUNC, in modo da selezionare una funzione alla prima chiamata, un'altra alla successiva, e così via.

Esecuzione del programma

Scrivete il programma della Figura 8.6 e provatelo; la prima approssimazione per la radice quadrata di 2 è 2; il valore corretto si ottiene dopo parecchie iterazioni. I risultati si presenteranno come in Figura 8.7.

```
x= 2.00000, fx= 2.00000,     dfx= 4.00000
x= 1.50000, fx= 2.50000E-1, dfx= 3.00000
x= 1.41667, fx= 6.94442E-3, dfx= 2.83333
x= 1.41422, fx= 5.96046E-6, dfx= 2.82843
x= 1.41421, fx=-1.19209E-7, dfx= 2.82843

La soluzione è   1.41421
```

Figura 8.7 — La radice positiva di $f(x) = x^2 - 2$.

Nel prossimo paragrafo apporteremo diverse piccole modifiche per rifinire questo programma; la prima ci consentirà di inserire diversi valori di prima approssimazione per la soluzione, cosa importante per studiare equazioni con più di una radice.

Input fornito dall'utente per la prima approssimazione

Quando la prima versione del metodo di Newton funziona correttamente, possiamo cominciare ad aggiungere altre prestazioni; nelle prossime versioni di questo capitolo saranno previsti sia l'ingresso che l'uscita su console. Perciò, la prima linea del programma sorgente diventerà:

```
PROGRAM NEWDR2 (INPUT, OUTPUT);
```

Fate una copia della prima versione, e modificate il programma principale (la parte nell'ultimo blocco BEGIN/END) secondo il listing di Figura 8.8.

```
BEGIN (* programma principale *)
  ALLDONE := FALSE;
  REPEAT
    WRITELN;
    WRITE ('Prima supposizione');
    READLN( X);
    IF X < -19.0 THEN ALLDONE := TRUE
    ELSE
      BEGIN
        NEWTON(X);
        WRITELN;
        WRITELN ('La soluzione è', X);
        WRITELN
      END
  UNTIL ALLDONE
END.
```

Figura 8.8 — Il programma principale della seconda versione.

Compilate la nuova versione e provatela. Nella prima versione abbiamo usato 2 come valore suggerito per la prima approssimazione; questa nuova versione è più sofisticata, e chiede all'utente di fornirgli in input il valore della prima approssimazione; quelle successive, insieme con i valori della funzione e delle sue derivate, vengono stampati come prima. Alla fine il programma ricomincia da capo e chiede all'utente di inserire un altro valore per la prima approssimazione.

Esecuzione del programma per determinare la seconda radice

Cominciate con il valore 2; i risultati dovrebbero essere gli stessi della prima versione. La seconda volta provate con 1; la prima approssimazione è dall'altra parte della radice, ma la radice quadrata di 2 dovrebbe essere calcolata con poche iterazioni. La terza volta inserite −2; notate che il procedimento di iterazione converge su di un'altra radice.

Sappiamo che l'equazione

$$x^2 - 2 = 0$$

ha due soluzioni; dando un valore negativo per la prima approssimazione abbiamo trovato l'altra radice.

Studiate che cosa succede quando la prima approssimazione è vicina al valore di mezzo tra le due radici. Provate con 0.0001; in pochi passi avrete la risposta. Infine provate con 0; la curva

$$y = f(x)$$

ha pendenza zero in questo punto, e possono succedere due cose: o si ha un errore di divisione in floating point, o il programma entra in un ciclo senza fine. Questa situazione sarà corretta nella prossima versione.

Un test per pendenza zero

Quando la derivata, o la pendenza della nostra funzione è zero, l'ultimo termine dell'equazione 4 diventa infinito. Stiamo cercando il punto in cui la tangente incontra l'asse delle x, ma come si vede dalla Figura 8.9 le due linee sono parallele, e quindi non hanno intersezione.

Aggiungeremo ora alcune istruzioni per controllare la pendenza della curva nel nostro programma. Un modo per far ciò è di definire un numero molto piccolo, come:

```
CONST SMALL = 1.0E -16;
```

La pendenza può essere controllata con l'istruzione:

```
IF ABS(DFX) <SMALL THEN...
```

Naturalmente il valore assegnato alla SMALL deve essere adeguato alla versione del Pascal che stiamo usando, deve cioè essere scelto con cura.

Il controllo che useremo noi è più immediato:

```
IF DFX = 0.0 THEN...
```

Se la pendenza è zero, viene stampato un messaggio di errore e viene settato un

flag; altrimenti il processo continua normalmente. Una seconda modifica deve essere fatta nel programma principale: spostate di due spazi a destra la linea

```
WRITELN ('La soluzione è', X);
```

che si trova verso la fine, e inserite immediatamente prima:

```
IF NOT ERROR THEN
```

Le due linee si presenteranno così:

```
IF NOT ERROR THEN
  WRITELN ('La soluzione è', X);
```

Lo scopo di questa nuova istruzione è di controllare il flag di errore dopo ogni chiamata alla procedura NEWTON; se il flag è resettato non c'è errore, e viene stampata la soluzione. Modificate la procedura NEWTON secondo la Figura 8.10; compilate la nuova versione e provatela.

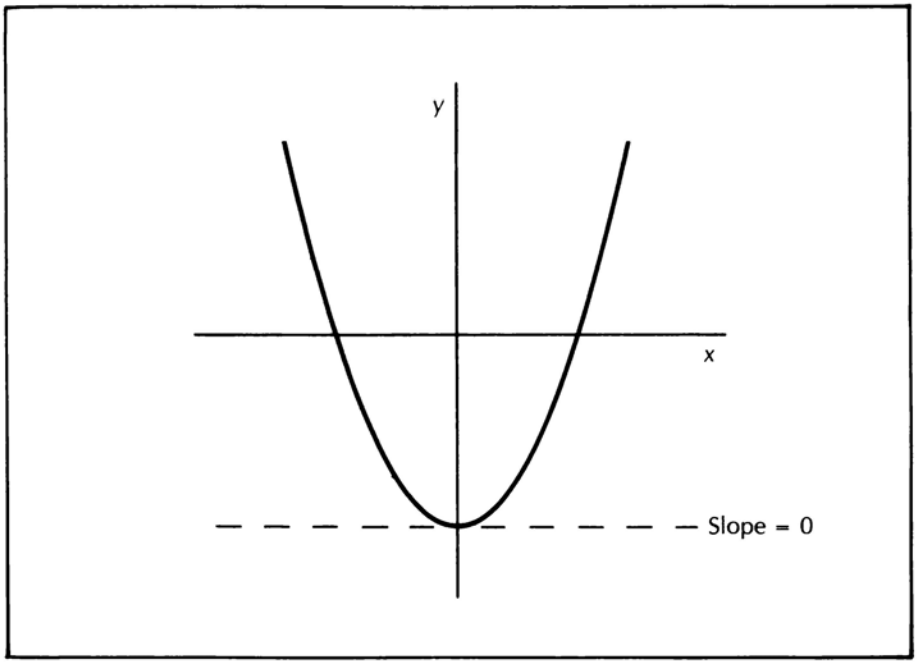


Figura 8.9 — Dove f'(x) = 0 la tangente è parallela all'asse x.

```
PROCEDURE NEWTON
            (VAR X: REAL);
 (* 23 Gen 81 *)

CONST
   TOL = 1.0E-6;

VAR
   FX, DFX, DX, X1 : REAL;

BEGIN (* newton *)
   ERROR := FALSE;
   REPEAT
      X1 := X;
      FUNC(X, FX, DFX);
      IF DFX = 0.0 THEN
         BEGIN
            ERROR := TRUE;
            X := 1.0;
            WRITELN('ERRORE: pendenza zero')
         END
      ELSE
         BEGIN
            DX := FX/DFX;
            X := X1 - DX;
            WRITELN
               ('x= ', X1, ', fx= ', FX, ', dfx= ', DFX)
         END
   UNTIL
      ERROR OR
      (ABS(DX) <= ABS(TOL * X))
END (* newton *);
```

Figura 8.10 — Il metodo di Newton con un test per tangente 0.

Esecuzione del programma con controllo della pendenza

Date come valori iniziali 2,1 e −1 come prima, per vedere se il programma funziona bene; poi inserite zero, valore che prima provocava un errore di divisione in floating point, e che dovrebbe ora essere trattato senza difficoltà. Il programma dovrebbe stampare un opportuno messaggio di errore, e chiedere un altro valore come prima approssimazione. Per uscire dal programma si deve inserire un valore iniziale inferiore a −19.

L'ultima cosa che vogliamo ottenere è che il programma finisca, stampando un messaggio di errore, quando, dopo un certo numero di iterazioni, non converge su una radice.

Quando il procedimento non converge

In certi casi, il metodo di Newton non converge su una radice, dopo un ragionevole numero di iterazioni, ad esempio quando le approssimazioni successive oscillano intorno a una radice complessa, come nella Figura 8.11.

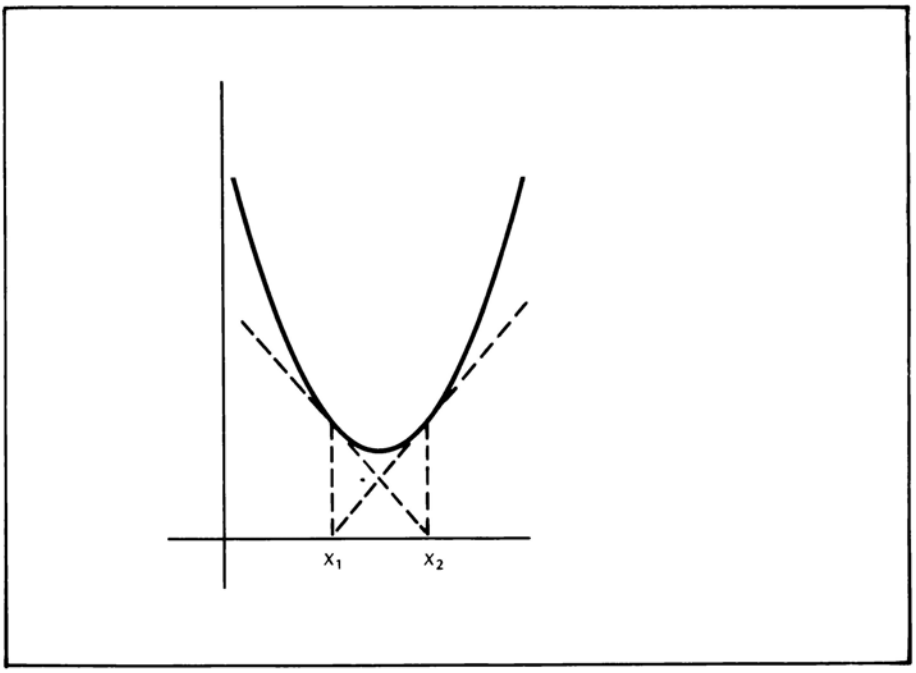


Figura 8.11 — Una radice complessa.

La prima approssimazione, x_1, dà origine a un secondo valore, x_2, che riproduce ancora x_1 come terza approssimazione, e il procedimento non finisce più.

Oppure può capitare che un'approssimazione sia molto lontana da una radice, anche se il primo valore è vicino a una radice. Possiamo osservare questo comportamento con la versione precedente del metodo di Newton, dando come valore della prima approssimazione 0.00001. La seconda approssimazione ha un valore di 20.000 che è decisamente fuori dai limiti; ogni successiva approssimazione è poi la metà di quella precedente. In questo caso si può anche trovare una soluzione, ma dopo più di venti iterazioni.

Per proteggerci dall'eventualità che il procedimento non converga su una soluzione, possiamo aggiungere un contatore di ciclo nella procedura NEWTON, in modo da interromperla se la convergenza non si verifica dopo un certo numero di iterazioni, ad esempio 20. In questo modo il programma è in grado di gestire sia il caso delle oscillazioni, sia quello di un'approssimazione molto lontana da una radice.

Modificate la procedura NEWTON secondo la Figura 8.12, compilate la nuova versione e provatela. Date una prima approssimazione di 500000, che è talmente lontana dalla radice da richiedere moltissimi cicli per convergere. Dopo 20 cicli il processo di iterazione finisce, e viene inviato un opportuno messaggio di errore. Provate poi con una prima approssimazione di 2 per assicurarvi che il programma funzioni correttamente.

Abbiamo dunque sviluppato e perfezionato il nostro programma; possiamo ora usarlo per trovare le radici di equazioni più complesse.

PROGRAMMI PASCAL: RISOLUZIONE DI ALTRE EQUAZIONI

Consideriamo l'equazione non lineare:

$$e^x = 4x$$

A differenza della precedente, questa non si può risolvere esplicitamente, mentre si presta proprio a essere risolta con il metodo di Newton. La corrispondente funzione e le sue derivate sono:

$$f(x) = e^x - 4x$$

$$f'(x) = e^x - 4$$

```
PROCEDURE NEWTON
            (VAR X: REAL);
(* 23 Gen 81 *)

CONST
   TOL = 1.0E-6;
   MAX = 20;

VAR
   FX, DFX, DX, X1 : REAL;
   I : INTEGER;

BEGIN (* newton *)
   ERROR := FALSE;
   I := 0;
   REPEAT
      I := I + 1;
      X1 := X;
      FUNC(X, FX, DFX);
      IF DFX = 0.0 THEN
         BEGIN
            ERROR := TRUE;
            X := 1.0;
            WRITELN ('ERRORE: pendenza zero')
         END
      ELSE
         BEGIN
            DX := FX/DFX;
            X := X1 - DX;
            WRITELN
               ('x= ', X1, ', fx= ', FX, ', dfx= ', DFX)
         END
```

Figura 8.12 — Metodo di Newton con un contatore di ciclo.

```
    UNTIL
       ERROR OR
       (I > MAX) OR
       (ABS(DX) <= ABS(TOL * X));
    IF I > MAX THEN
       BEGIN
          WRITELN
                ('ERRORE: non convergente in',
                MAX, 'cicli');
          ERROR := TRUE
       END
  END (* newton *);
```

Figura 8.12 — Metodo di Newton con un contatore di ciclo (segue).

Cambiate la procedura FUNC secondo la figura 8.13. Notate che vi è stata introdotta la variabile E per non calcolare due volte e^X. Ricordate che

$$\frac{de^x}{dx} = e^x$$

```
  PROCEDURE FUNC(X : REAL;
          VAR FX, DFX : REAL);
  VAR
    E : REAL;
  BEGIN
    E   := EXP(X);
    FX  := E - 4.0 * X;
    DFX := E - 4.0
  END (* func *);
```

Figura 8.13 — Procedura FUNC per $e^x = 4x$.

Eseguite la nuova versione con una prima approssimazione di 4. Il programma dovrebbe convergere su un valore di 2.153, che è una delle due radici. L'altra radice si trova dando una prima approssimazione di 0.1, e la soluzione, in questo caso, è 0.357.

La prossima equazione contiene la funzione SIN, e quindi deve essere risolta in due modi diversi, a seconda del compilatore Pascal a disposizione.

Una funzione con molte radici

Studiamo le diverse radici dell'equazione:

$$\sin(x) = \frac{x}{10}$$

Modificate la procedura FUNC secondo la Figura 8.14.

Una delle radici della nuova equazione è zero; man mano che ci avviciniamo a questa radice, dobbiamo calcolare il SIN di numeri sempre più piccoli, e, come abbiamo visto nel Capitolo 1, molti compilatori Pascal per microprocessori usano un procedimento che crea dei problemi in questi casi. Più precisamente, il problema è che l'argomento del SIN viene elevato al quadrato prima di effettuare un controllo sull'intervallo.

Se non avete ancora eseguito il programma di test del Capitolo 1, dovete farlo adesso. Se i risultati di questi test indicano che l'intervallo di applicazione della funzione SIN è lo stesso delle altre operazioni in floating-point, probabilmente potete usare la procedura FUNC della Figura 8.14. Se invece l'intervallo valido per la funzione SIN è circa la radice quadrata di quello delle operazioni in floating-point, allora dovete usare la procedura FUNC della Figura 8.15.

```
PROCEDURE FUNC(X : REAL;
        VAR FX, DFX : REAL);

BEGIN
  FX  := SIN( X ) - 0.1 * X;
  DFX := COS( X ) - 0.1
END (* func *);
```

Figura 8.14 — Soluzione di sin(x) = x/10.

Controlli effettuati su diversi compilatori pascal hanno dato i seguenti valori:

	Sin	*General*
Pascal 1	10^{-38}	10^{-38}
Pascal 2	10^{-18}	10^{-18}
Pascal 3	10^{-18}	10^{-38}
Pascal 4	10^{-21}	10^{-64}
Pascal 5	10^{-9}	10^{-18}

La procedura di Figura 8.15 effettua un controllo sull'argomento del seno o del coseno; se è più piccolo di SMALL, non vengono chiamate le funzioni SIN e COS di sistema, e viene assegnato il valore dell'argomento al seno, e il valore 1 al coseno. Il valore di SMALL dipende dal compilatore Pascal che usate; se siete in dubbio, usate il valore della Figura 8.15.

```
PROCEDURE FUNC(X : REAL;
        VAR FX, DFX : REAL);
CONST
  SMALL = 1.0E-8;
VAR
  S, C : REAL;
BEGIN
  IF ABS(X) > SMALL THEN
    BEGIN
      S := SIN( X );
      C := COS( X )
    END
  ELSE
    BEGIN
      S := X;
      C := 1.0
    END;
  FX  := S - 0.1 * X;
  DFX := C - 0.1
END (* func *);
```

Figura 8.15 — Procedura alternativa FUNC per sin(x) = x/10.

Compilate quest'ultima versione e provatela. Date una prima approssimazione di 1.0, che dovrebbe far convergere il programma sulla radice zero. Poi provate con –1.0; anche in questo caso il programma dovrebbe convergere a zero. Questa equazione ha molte altre radici; la tabella di figura 8.16 dà le prime approssimazioni con le radici corrispondenti. Provatele tutte per verificare che il programma funzioni correttamente.

1ª approssimazione	*radice*
–1	0
1	0
4	2.852..
4.3	7.068..
4.5	0
4.7	–8.423..
5	–2.852..
6	7.068..
9	8.423..

Figura 8.16 — Le radici di f(x) = sin x – x/10.

PROGRAMMA PASCAL: L'EQUAZIONE DELLA PRESSIONE DI UN VAPORE

Siamo in grado di risolvere l'equazione della pressione di un vapore introdotta all'inizio del capitolo. Scriviamo la funzione e la sua derivata:

$$f(T) = A + \frac{B}{T} + C \ln T - \ln P$$

$$df(T) = \frac{-B}{T^2} + \frac{C}{T}$$

Ricordate che A, B, C e P sono costanti. Fate una copia del programma di Newton e modificate la procedura FUNC secondo la Figura 8.17. Notate che si usa la lettera t al posto di X; questo è possibile perché i parametri della procedura sono "dummy".

```
PROCEDURE FUNC(T : REAL;
        VAR FT, DFT : REAL);

(* la pressione del vapore di conduzione *)

CONST
  A = 18.19;
  B = -23180.0;
  C = -0.8858;
  LOGP = -4.60517 (* ln(.01) *);

BEGIN
  FT := A + B/T + C * LN(T) - LOGP;
  DFT := -B/(T * T) + C/T
END (* func *);
```

Figura 8.17 — Soluzione dell'equazione della pressione del vapore.

Con questa versione della FUNC troveremo la temperatura corrispondente a una pressione di vapore di 0.01 atmosfere; il logaritmo di questa pressione, —4.60517 viene introdotto direttamente come costante.

Compilate la nuova versione del programma e provatela. Date una prima approssimazione di 500, e il programma convergerà a una temperatura di 1416 in circa sette cicli.

SOMMARIO

Abbiamo visto come è facile scrivere un programma che applica il metodo di Newton per trovare le radici di un'equazione, dapprima usando una funzione particolarmente semplice, e poi perfezionando il programma per funzioni più complesse, come esponenziali e funzioni trigonometriche. Abbiamo infine risolto l'equazione di un'applicazione scientifica reale.

CAPITOLO 9

INTEGRAZIONE NUMERICA

INTRODUZIONE

In questo capitolo svilupperemo tre diversi metodi per eseguire un'integrazione numerica, cioè per determinare l'area della figura sottesa da una curva, nell'intervallo compreso tra due valori dati della variabile indipendente. Scriveremo un programma Pascal per ognuno di questi metodi, e confronteremo la loro efficienza. Tutti e tre questi metodi usano figure con area misurabile, di dimensioni sempre più piccole, che suddividono la figura di cui si deve calcolare l'area; la somma delle loro aree dà un valore approssimato dell'area della figura.

Nel metodo trapezoidale, il lato superiore delle figure si ottiene da rette secanti la curva, mentre nel metodo di Simpson si ottiene da parabole. Entrambi questi metodi possono essere perfezionati calcolando l'errore con uno sviluppo in serie di Taylor (metodo di correzione finale).

Il terzo metodo, quello di Romberg, è il più complesso; costruisce una matrice di interpolazioni per arrivare all'area totale. Una delle funzioni a cui applicheremo entrambi i metodi, di Simpson e di Romberg, assomiglia alla funzione di distribuzione normale, su cui torneremo nel Capitolo 11. Infine, studieremo un metodo per integrare una funzione che tende all'infinito in uno dei limiti dell'intervallo.

Cominciamo con una breve descrizione dell'integrale definito e dei metodi per calcolarlo.

L'INTEGRALE DEFINITO

Il valore dell'integrale definito:

$$\int_a^b f(x)dx = F(b) - F(a)$$

dove $F'(x) = dF(x)/dx = f(x)$, può essere interpretato come misura dell'area della figura sottesa dalla curva $f(x)$, nell'intervallo a-b, come illustrato nella figura 9.1.

Il calcolo dell'integrale è immediato, in certi casi, mentre è molto difficile in altri. Per esempio, la serie di potenze:

$$1 + 2x + 3x^2$$

può essere integrata termine a termine nell'intervallo a,b, e si ottiene:

$$b + b^2 + b^3 - a - a^2 - a^3$$

Funzioni più complesse possono essere integrate facendo uso delle tavole di integrazione che si trovano nei libri di matematica; talvolta si può usare la tecnica di integrazione per parti, oppure sostituire la funzione originale con una serie o un'espansione asintotica, e integrare questa funzione.

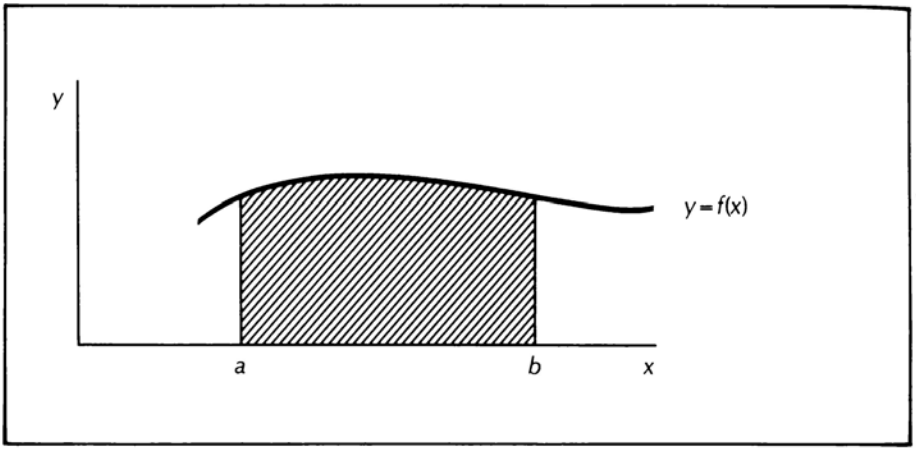


Figura 9.1 — L'area sottesa dalla curva $y = f(x)$.

Può capitare che l'integrando non sia una vera e propria funzione, ma un insieme di dati sperimentali; in questo caso è possibile interpolare i dati con una funzione che viene poi integrata. Ci sono dei casi, però, in cui risulta impossibile o molto difficile trovare un'espressione analitica dell'integrando; se sono noti i limiti, una soluzione accettabile si può allora trovare con metodi di approssimazione. Questo approccio prende il nome di integrazione numerica.

Esistono diversi metodi di integrazione numerica, che comportano in genere la sostituzione della funzione originale con una facilmente integrabile, ad esempio una funzione polinomiale come una retta o una parabola, o una funzione trascendente

come il seno o il coseno. La precisione del risultato dipende da come la nuova funzione approssima l'originale.

Nel prossimo paragrafo descriveremo i due metodi generali di sostituzione, e una semplice implementazione particolare della sostituzione con una retta.

LA REGOLA TRAPEZOIDALE

Uno dei metodi più semplici di integrazione numerica è quella nota come la regola trapezoidale. In questo metodo, la funzione originale è approssimata da un insieme di rette. La regione di spazio da integrare è suddivisa in tanti trapezi uguali, di larghezza

$$\Delta x = \frac{b - a}{n}$$

dove n è il numero dei trapezi, e a e b sono gli estremi dell'intervallo di integrazione. Se la curva viene interpolata con una sola linea retta, cioè se c'è un solo trapezio (come in figura 9.2), l'area calcolata risulta:

$$\frac{(b - a)[f(a) + f(b)]}{2}$$

dove f(a) è il valore della funzione all'estremo sinistro, e f(b) all'estremo destro.

Nel caso più generale, la figura viene suddivisa in n trapezi; un trapezio interno generico è limitato a sinistra da una retta verticale passante per x_i, e a destra da una retta verticale passante per $x_i + \Delta x$. Il lato inferiore del trapezio coincide con l'asse delle x, e la curva originale è sostituita con una linea retta, che in genere ha una tangente diversa da zero, cioè non è parallela all'asse delle x. La Figura che ne risulta ha la forma di un trapezio, da cui il nome del metodo.

I trapezi possono essere numerati da 1 a n; a noi però interessa il valore della funzione all'estremo sinistro e a quello destro di ogni trapezio. Per n trapezi ci sono n − 1 lati interni alla figura, più i due corrispondenti agli estremi dell'intervallo di integrazione, per un totale di n+1 lati, numerati da 0 a n.

L'area del primo trapezio è:

$$\frac{[f(0) + f(1)]\Delta x}{2}$$

o

$$\frac{[f(a) + f(1)]\Delta x}{2}$$

e l'area dell'ultimo trapezio è:

$$\frac{[f(n-1) + f(n)]\Delta x}{2}$$

o

$$\frac{[f(n-1) + f(b)]\Delta x}{2}$$

L'area del trapezio i-esimo è:

$$\frac{[f(i-1) + f(i)]\Delta x}{2}$$

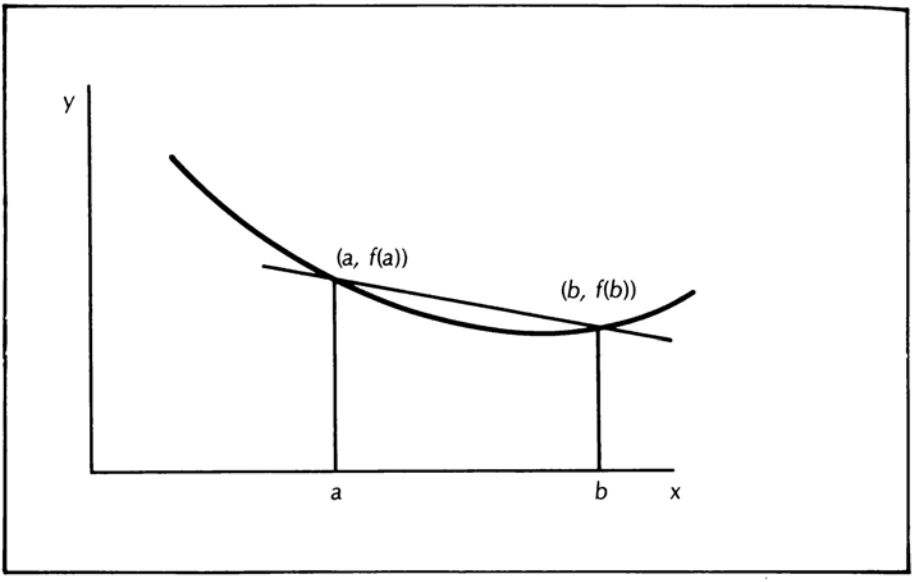


Figura 9.2 — Calcolo dell'area con una sola sezione.

dove f(i—1) è il valore della funzione all'estremo sinistro dell'i-esimo trapezio, e f(i) è il valore della funzione all'estremo destro. Il valore dell'integrale è la somma delle aree di tutti i trapezi, secondo l'espressione:

$$\{[f(a) + f(1)] + [f(1) + f(2)] + [f(2) + f(3)] + \dots$$
$$+ [f(n-2) + f(n-1)] + [f(n-1) + f(b)]\} \frac{\Delta x}{2}$$

Il lato destro del primo trapezio è il sinistro del secondo, così come il lato sinistro dell'ultimo trapezio è il destro del penultimo. I lati di tutti gli altri trapezi sono in co-

mune a due di essi. La formula di integrazione con la regola trapezoidale può dunque essere semplificata come segue:

$$[f(a) + 2f(1) + 2f(2) + \ldots + 2f(n-2) + 2f(n-1) + f(b)] \frac{\Delta x}{2}$$

Il nostro primo programma su questo metodo ci consentirà di provare diverse suddivisioni dell'area totale, dando di volta in volta un diverso numero di trapezi.

PROGRAMMA PASCAL: LA REGOLA TRAPEZOIDALE CON IL NUMERO DI TRAPEZI INTRODOTTO COME DATO DI INGRESSO DALL'UTENTE

L'area calcolata con la regola trapezoidale approssima sempre meglio il valore effettivo dell'area al crescere del numero di trapezi, come dimostra il programma della Figura 9.3. Scrivete il programma ed eseguitelo. Il programma principale chiede all'utente il numero delle sezioni in cui deve essere suddivisa la figura, e da questo dipende il risultato.

L'algoritmo di integrazione è contenuto nella procedura TRAPEZ, e la funzione da integrare

$$\int_1^9 \frac{dx}{x}$$

è definita dalla funzione FX, che è una routine separata dalla TRAPEZ. Poichè questa funzione può essere facilmente integrata, possiamo confrontare il valore esatto dell'integrale con quello calcolato con la regola trapezoidale.

```
PROGRAM TRAP1(INPUT, OUTPUT);

(* integrazione per mezzo della regola dei trapezi *)
(* 24 Dic 80 *)

VAR
   DONE : BOOLEAN;
   SUM, UPPER, LOWER : REAL;
   PIECES : INTEGER;
```

Figura 9.3 — La regola trapezoidale.

```
FUNCTION FX(X : REAL) : REAL;

(* dichiara f(x) = 1/x *)
(* si badi ad x = 0 *)

BEGIN
  FX := 1.0/X
END;

PROCEDURE TRAPEZ(LOWER, UPPER : REAL;
                 PIECES  : INTEGER;
                 VAR SUM  : REAL);

(* integrazione numerica con il metodo dei trapezi *)
(* la funzione è FX, gli estremi sono LOWER ed UPPER *)
(* carica numero di vettori uguale a PIECES *)
(* la partizione fissata è DELTA_X, il risultato è SUM *)

VAR
  I : INTEGER;
  X, DELTA_X, ESUM, PSUM : REAL;

 BEGIN
   DELTA_X := (UPPER - LOWER)/PIECES;
   ESUM := FX(LOWER) + FX(UPPER);
   PSUM := 0.0;

   FOR I := 1 TO PIECES - 1 DO
     BEGIN
       X := LOWER + I * DELTA_X;
       PSUM := PSUM + FX(X)
     END;
```

Figura 9.3 — La regola trapezoidale (segue).

```
    SUM := (ESUM + 2.0 * PSUM) * DELTA_X * 0.5
END; (* trapez *)

BEGIN (* programma principale *)
    DONE := FALSE;
    LOWER := 1.0;
    UPPER := 9.0;
    WRITELN;
    REPEAT
        WRITE('Quante sezioni?');
        READLN(PIECES);
        IF PIECES < 0 THEN DONE := TRUE

        ELSE
            BEGIN
                TRAPEZ(LOWER, UPPER, PIECES, SUM);
                WRITELN(' area = ', SUM)
            END
    UNTIL DONE
END.
```

Figura 9.3 — La regola trapezoidale (segue).

Il valore dell'integrale è il logaritmo naturale di 9, cioè 2.197225.
Studieremo ora una versione più sofisticata del programma.

PROGRAMMA PASCAL: UNA REGOLA TRAPEZOIDALE PERFEZIONATA

Possiamo migliorare il programma in due modi: prima di tutto possiamo modificare la procedura TRAPEZ in modo che divida automaticamente la figura originale in un numero sempre maggiore di sezioni, e poi possiamo evitare una gran quantità di calcoli usando ogni volta i risultati del ciclo precedente.

Modificate il programma secondo la Figura 9.4, compilatelo ed eseguitelo.

```
PROGRAM TRAP2(OUTPUT);

(* integrazione per mezzo della regola dei trapezi *)
(* 6 Feb 81 *)

CONST
   TOL = 1.0E-5;

VAR
   SUM, UPPER, LOWER : REAL;

FUNCTION FX(X : REAL) : REAL;

(* dichiara f(x) = 1/x *)
(* si badi ad x = 0 *)

BEGIN
   FX := 1.0/X
END;

PROCEDURE TRAPEZ(LOWER, UPPER, TOL : REAL;
                                VAR SUM : REAL);

(* integrazione numerica con il metodo dei trapezi *)
(* la funzione esterna è FX *)
(* gli estremi sono LOWER ed UPPER *)
(* con numero di regioni uguale a PIECES *)
(* l'estensione delle partizioni è DELTA_X, il risultato è SUM *)

VAR
   PIECES, I: INTEGER;
   X, DELTA_X, END_SUM, MID_SUM, SUM1: REAL;
```

Figura 9.4 — Un metodo trapezoidale perfezionato.

```
BEGIN
  PIECES := 1;
  DELTA_X := (UPPER - LOWER)/PIECES;
  END_SUM := FX(LOWER) + FX(UPPER);
  SUM := END_SUM * DELTA_X/2.0;
  WRITELN('   1', SUM);
  MID_SUM := 0.0;
  REPEAT
    PIECES := PIECES * 2;
    SUM1 := SUM;
    DELTA_X := (UPPER - LOWER)/PIECES;
    FOR I := 1 TO PIECES DIV 2 DO
      BEGIN
        X := LOWER + DELTA_X * (2.0 * I - 1.0);
        MID_SUM := MID_SUM + FX(X)
      END;
    SUM := (END_SUM + 2.0 * MID_SUM) * DELTA_X * 0.5;
    WRITELN(PIECES:5, SUM)
  UNTIL ABS(SUM - SUM1) <= ABS(TOL * SUM)
END; (* trapez *)


BEGIN (* programma principale *)
  LOWER := 1.0;
  UPPER := 9.0
  WRITELN;
  TRAPEZ(LOWER, UPPER, TOL, SUM);
  WRITELN;
  WRITELN('   area = ', SUM)
END.
```

Figura 9.4 — Un metodo trapezoidale perfezionato (segue).

Esecuzione del programma

La prima volta, tutta la figura è considerata come un unico trapezio; il numero di sezioni viene poi raddoppiato, e viene calcolata nuovamente l'area. La procedura continua raddoppiando ogni volta il numero di suddivisioni; all'aumentare di questo aumenta la precisione del risultato, e, naturalmente, il tempo di esecuzione. Ad ogni ciclo vengono stampati il numero delle sezioni e il corrispondente valore dell'area; l'uscita si presenta come in Figura 9.5.

Dalla Figura 9.5 si vede che il valore calcolato è tanto più vicino al valore esatto dell'integrale quanto maggiore è il numero di suddivisioni. Il procedimento ha termine quando due valori consecutivi sono all'interno della tolleranza desiderata. La tolleranza è una variabile globale a cui viene assegnato il valore 10^{-5} all'inizio del programma; potete cambiare questo valore in base alla precisione del vostro compilatore Pascal; volendo, potete anche togliere dalla procedura TRAPEZ l'istruzione che stampa i risultati di ogni iterazione.

```
   1 4.44444
   2 3.02222
   4 2.46349
   8 2.27341
  16 2.21733
  32 2.20234
  64 2.19851
 128 2.19755
 256 2.19731
 512 2.19725
1024 2.19723

area =  2.19723
```

Figura 9.5 — Integrazione con il metodo trapezoidale perfezionato.

In questa versione, il numero di suddivisioni ad ogni iterazione è il doppio del precedente. Notate che non è necessario ricalcolare le misure di ogni lato ad ogni iterazione; supponiamo ad esempio di usare a un certo punto quattro trapezi; il ciclo successivo ne userà 8. Poichè i lati dei trapezi del ciclo precedente sono comuni a quelli del ciclo attuale, è più rapido conservare la loro somma e utilizzarla nel ciclo attuale piuttosto che ricalcolare tutte le misure. Devono essere calcolate solo le misure dei lati corrispondenti ai punti di mezzo tra due della precedente suddivisione, per poi sommarle alla precedente somma dei valori interni. La nuova somma è poi molti-

plicata per 2 e aggiunta ai valori corrispondenti agli estremi dell'intervallo di integrazione; infine il risultato è moltiplicato per 2.

Questa versione è più veloce della precedente, ma presenta gli stessi errori. Il problema è che cerchiamo di interpolare una curva con un insieme di linee rette. Se la curva originale non è essa stessa una retta, ci vorranno molte iterazioni prima di convergere su un buon risultato.

Nell'ultima versione della regola trapezoidale aggiungeremo una procedura di "correzione finale" al nostro programma.

Tecniche di correzione finale

Possiamo migliorare ulteriormente il nostro programma con dei fattori di correzione che si ottengono da uno sviluppo in serie di Taylor. Questi termini contengono la derivata seconda della funzione, ma fortunatamente si possono usare al loro posto dei termini semplificati. I termini relativi ai punti intermedi scompaiono, lasciando solo quelli relativi agli estremi inferiore e superiore dell'intervallo. Così il termine di correzione maggiore contiene la derivata della funzione calcolata agli estremi x_0 e x_n. La quantità da aggiungere alla somma è

$$\frac{[f'(b) - f'(a)](\Delta x)^2}{12}$$

dove f'(b) è il valore della tangente in b e f'(a) è il valore della tangente in a. Il valore risultante è sottratto dalla somma trapezoidale. Questo metodo è noto come la regola trapezoidale con correzione finale.

PROGRAMMA PASCAL: REGOLA TRAPEZOIDALE CON CORREZIONE FINALE

Fate una copia del programma precedente. Aggiungete la funzione DFX (Figura 9.6) subito dopo la funzione FX. Questa procedura calcola la derivata della funzione. Avremmo potuto naturalmente calcolare la funzione e la sua derivata in una stessa routine, ma avremmo dovuto usare una procedura invece che una funzione. Inoltre la derivata va calcolata solo agli estremi. Modificate la procedura TRAPEZ secondo la figura 9.6.

```
FUNCTION DFX(X : REAL) : REAL;

(* dichiara la derivata di f(x) = 1/x *)

BEGIN
   DFX := -1.0/SQR(X)
END;

PROCEDURE TRAPEZ(LOWER, UPPER, TOL : REAL;
                                 VAR SUM : REAL);

(* integrazione numerica con il metodo dei trapezi *)
(* la funzione esterna è FX *)
(* gli estremi sono LOWER ed UPPER *)
(* con un numero di regioni uguale a PIECES *)
(* la misura delle partizioni è DELTA_X, il risultato è SUM *)

VAR
   PIECES,I : INTEGER;
   X, DELTA_X, END_SUM, MID_SUM,
   END_COR, SUM1: REAL;


BEGIN
   PIECES := 1;
   DELTA_X := (UPPER - LOWER)/PIECES;
   END_SUM := FX(LOWER) + FX(UPPER);
   END_COR := (DFX(UPPER) - DFX(LOWER))/12.0;
   SUM := END_SUM * DELTA_X/2.0;
   WRITELN('  1', SUM);
   MID_SUM := 0.0;
```

Figura 9.6 — Funzione DFX e procedura TRAPEZ con correzione finale.

```
    REPEAT
      PIECES := PIECES * 2;
      SUM1 := SUM;
      DELTA_X := (UPPER - LOWER)/PIECES;
      FOR I := 1 TO PIECES DIV 2 DO
        BEGIN
          X := LOWER + DELTA_X * (2.0 * I - 1.0);
          MID_SUM := MID_SUM + FX(X)
        END;
      SUM := (END_SUM + 2.0 * MID_SUM) * DELTA_X
            * 0.5 - SQR(DELTA_X) * END_COR;
      WRITELN(PIECES:5, SUM)
    UNTIL ABS(SUM - SUM1) <= ABS(TOL * SUM)
  END (* trapez *);
```

Figura 9.6 — Funzione DFX e procedura TRAPEZ con correzione finale (segue).

Esecuzione del programma

Compilate il programma ed eseguitelo; i risultati si presenteranno come in Figura 9.7.

Notate come il procedimento converge molto più rapidamente in questo caso; no-

```
  1 4.44444
  2 1.70535
  4 2.13427
  8 2.19111
 16 2.19675
 32 2.19719
 64 2.19722
128 2.19723

area =  2.19723
```

Figura 9.7 — Integrazione trapezoidale con correzione finale.

tate anche che durante il processo di iterazione, la somma supera il valore esatto e poi inverte la sua direzione. Dovrebbe essere chiaro, d'altra parte, che non è sempre possibile usare la tecnica di correzione finale con la regola trapezoidale, ad esempio quando la tangente in un estremo è infinita o molto grande; discuteremo questo caso alla fine del capitolo. Inoltre, se la funzione ha tangente zero agli estremi, il termine di correzione è nullo.

La regola trapezoidale sostituisce la funzione con una polinomiale del primo ordine (una retta). In molti casi è facile immaginare che una funzione polinomiale del secondo ordine (una parabola) andrebbe meglio; su questa ipotesi si basa il metodo che implementeremo con il prossimo programma. Studieremo tre diverse esecuzioni del programma, la prima con la stessa funzione trattata dal programma della regola trapezoidale, poi con un'esponenziale e infine con la funzione seno.

PROGRAMMA PASCAL: METODO DI INTEGRAZIONE DI SIMPSON

Il metodo di integrazione numerica di Simpson è simile a quello trapezoidale, ma la curva originale è sostituita da un insieme di parabole invece che di rette. Poichè una parabola può essere definita da un minimo di tre punti, dobbiamo dividere la figura originale in un numero pari di sezioni. Ogni parabola interpola la curva su due sezioni adiacenti. In genere ci si può aspettare che le parabole interpolino la curva con miglior approssimazione delle rette, e quindi sarà necessario un minor numero di suddivisioni per ottenere una buona convergenza. La formula è:

$$(f_0 + f_n + 4 \sum_{\substack{j=1 \\ j \text{ odd}}}^{n-1} f_j + 2 \sum_{\substack{j=2 \\ j \text{ even}}}^{n-2} f_j) \frac{\Delta x}{3}$$

Prima esecuzione del programma del metodo di Simpson

Scrivete il programma della Figura 9.8 ed eseguitelo; i risultati si presenteranno come in Figura 9.9.

Confrontate la Figura 9.9 con la 9.5 e la 9.7. In ogni punto, il valore dell'integrale ottenuto con la regola di Simpson è più vicino a quello esatto del valore ottenuto con la regola trapezoidale, a parità del numero di suddivisioni. Di conseguenza è necessario un minor numero di iterazioni e la convergenza si ottiene più rapidamente. D'altra parte, la convergenza è circa la stessa del metodo trapezoidale con correzione finale.

Ad ogni ciclo del metodo di Simpson, la funzione viene calcolata solo alle posizioni dispari. La somma relativa alle posizioni pari si ottiene da quella di tutte le posizioni intermedie del ciclo precedente, sia pari che dispari.

```
PROGRAM simp1(output);

(* integrazione con il metodo di Simpson *)
(* 24 Dic 80 *)

CONST
   TOL = 1.0E-5;

VAR
   SUM, UPPER, LOWER : REAL;

FUNCTION FX(X : REAL) : REAL;

(* dichiara f(x) = 1/x *)
(* si badi a x = 0 *)

BEGIN
   FX := 1.0/X
END (* function fx *);

PROCEDURE SIMPS(LOWER, UPPER, TOL : REAL;
                                VAR SUM : REAL);

(* integrazione numerica con la regola di Simpson *)
(* la funzione è FX, gli estremi sono LOWER ed UPPER *)
(* con un numero di regioni uguale a PIECES *)
(* la partizione è DELTA_X, il risultato è SUM *)

VAR
   I : INTEGER;
   X, DELTA_X, EVEN_SUM,
   ODD_SUM, END_SUM, SUM1 : REAL;
   PIECES : INTEGER;
```

Figura 9.8 — Regola di Simpson.

```
BEGIN
   PIECES := 2;
   DELTA_X := (UPPER - LOWER)/PIECES;
   ODD_SUM := FX(LOWER + DELTA_X);
   EVEN_SUM := 0.0;
   END_SUM := FX(LOWER) + FX(UPPER);
   SUM := (END_SUM + 4.0 * ODD_SUM) * DELTA_X/3.0;
   WRITELN(PIECES:5, SUM);
   REPEAT
      PIECES := PIECES * 2;
      SUM1 := SUM;
      DELTA_X := (UPPER - LOWER)/PIECES;
      EVEN_SUM := EVEN_SUM + ODD_SUM;
      ODD_SUM := 0.0;
      FOR I:= 1 TO PIECES DIV 2 DO
         BEGIN
            X := LOWER + DELTA_X * (2.0 * I - 1.0);
            ODD_SUM := ODD_SUM + FX(X)
         END;
      SUM := (END_SUM + 4.0 * ODD_SUM
            + 2.0 * EVEN_SUM) * DELTA_X/3.0;
      WRITELN(PIECES:5, SUM)
   UNTIL ABS(SUM - SUM1) <= ABS(TOL * SUM)
END (* simps *);

BEGIN (* programma principale *)
   LOWER := 1.0;
   UPPER := 9.0;
   WRITELN;
   SIMPS(LOWER, UPPER, TOL, SUM);
   WRITELN;
   WRITELN(' area = ', SUM)
END.
```

Figura 9.8 — Regola di Simpson (segue).

Seconda esecuzione del programma di Simpson
Una funzione esponenziale

Consideriamo un secondo esempio di integrazione con il metodo di Simpson. La funzione che studieremo assomiglia alla distribuzione normale descritta nel Capitolo 2, e anche alla funzione errore descritta nel Capitolo 11. Questa volta calcoleremo l'integrale tra 0 e 1.

Fate una copia del programma precedente e modificatelo in modo da trovare il valore dell'integrale:

$$\int_0^1 e^{-x^2}\,dx$$

Modificate la funzione FX come segue:

```
FUNCTION TX(X:REAL) : REAL;
BEGIN
  FX : = EXP(-X*X)
END;
```

```
  2 2.54815
  4 2.27725
  8 2.21005
 16 2.19864
 32 2.19734
 64 2.19723
128 2.19723

area =  2.19723
```

Figura 9.9 — Integrazione con la regola di Simpson.

Poi modificate gli estremi di integrazione nella prima e nella seconda linea del programma principale come segue:

```
LOWER: = 0.0;
UPPER: = 1.0;
```

Eseguite la nuova versione e confrontate i risultati con la Figura 9.10. Notate come il procedimento converge rapidamente.

```
 2    0.747180
 4    0.746855
 8    0.746826
16    0.746824

area =  0.746824
```

Figura 9.10 — Integrazione di e^{-x^2} con la regola di Simpson.

Terza esecuzione del programma di Simpson — Una funzione periodica

Le funzioni che abbiamo integrato finora hanno la curvatura nello stesso senso in tutto l'intervallo di integrazione. Consideriamo ora la funzione periodica:

$$\sin^2 x$$

nell'intervallo da 0 a 4π. Questa funzione è sempre positiva, e, di conseguenza, l'integrale in qualsiasi intervallo dovrebbe dare un risultato positivo; ha però valore zero agli estremi dell'intervallo, nel punto di mezzo e nei punti che si trovano a un quarto e a tre quarti dell'intervallo. Perciò, la prima area calcolata, sia con la regola trapezoidale che con il metodo di Simpson, sarà nulla. La successiva approssimazione, invece, con il metodo di Simpson darà un risultato positivo.

Fate una copia del programma precedente per calcolare questa funzione nell'intervallo tra 0 e 4π. Aggiungete l'istruzione:

```
PI = 3.141593;
```

nella dichiarazione delle costanti all'inizio del programma; poi modificate la funzione FX come segue:

```
BEGIN
  FX : = SQR(SIN(X))
END;
```

Infine modificate gli estremi dell'intervallo nelle prime linee del programma principale:

```
LOWER: = 0.0;
UPPER: = 4.0 * PI;
```

Eseguite il programma e confrontate i risultati con la Figura 9.11.

```
 2 8.57179E-12
 4 7.55984E-12
 8 8.37758
16 6.28319
32 6.28319

area =  6.28319
```

Figura 9.11 — L'integrale di $\sin^2 x$ tra 0 e 4π.

Se avessimo programmato questa routine in modo da finire quando il valore assoluto dell'area diventa inferiore a TOL, si sarebbe fermata dopo il primo ciclo, dando come risultato circa zero, invece che il valore esatto 6.28319. Si capisce quindi perchè abbiamo dovuto scegliere una tolleranza relativa come criterio di convergenza.

Ricaveremo ora una formula per la correzione finale per il metodo di Simpson.

PROGRAMMA PASCAL: IL METODO DI SIMPSON CON CORREZIONE FINALE

Le tecniche di correzione dell'errore si possono applicare al metodo di Simpson come a quello trapezoidale. Il termine principale contiene la derivata quarta della funzione, ma in questo caso è meglio usare un'approssimazione che richiede solo la derivata prima. Come per la regola trapezoidale, il calcolo della derivata è necessario solo agli estremi dell'intervallo. La formula del metodo di Simpson con correzione finale è la seguente:

$$\{7(f_0 + f_n) + 14 \sum_{\substack{j=2 \\ j \text{ even}}}^{n-2} f_j + 16 \sum_{\substack{j=1 \\ j \text{ odd}}}^{n-1} f_j + \Delta x[f'(a) - f'(b)]\} \frac{\Delta x}{15}$$

Esecuzione del programma con correzione finale

Fate una copia del programma di integrazione di Simpson della Figura 9.8. Aggiungete la funzione DFX che abbiamo usato nel metodo trapezoidale con correzione finale e modificate la procedura Simpson secondo la Figura 9.12.

```
PROCEDURE SIMPS(LOWER, UPPER, TOL : REAL;
                                VAR SUM : REAL);

(* integrazione numerica con la regola di Simpson *)
(* la funzione è FX, gli estremi sono LOWER e UPPER *)
(* con un numero di regioni uguale a PIECES *)
(* la partizione è DELTA_X, il risultato è SUM *)

VAR
   I : INTEGER;
   X, DELTA_X, EVEN_SUM,
   ODD_SUM, END_SUM,
   END_COR, SUM1 : REAL;
   PIECES : INTEGER;

BEGIN
   PIECES := 2;
   DELTA_X := (UPPER - LOWER)/PIECES;
   ODD_SUM := FX(LOWER + DELTA_X);
   EVEN_SUM := 0.0;
   END_SUM := FX(LOWER) + FX(UPPER);
   END_COR := DFX(LOWER) - DFX(UPPER);
   SUM := (END_SUM + 4.0 * ODD_SUM) * DELTA_X/3.0;
   WRITELN(PIECES:5, SUM);

   REPEAT
      PIECES := PIECES * 2;
      SUM1 := SUM;
      DELTA_X := (UPPER - LOWER)/PIECES;
      EVEN_SUM := EVEN_SUM + ODD_SUM;
      ODD_SUM := 0.0;
      FOR I:= 1 TO PIECES DIV 2 DO
```

Figura 9.12 — Procedura SIMPS con correzione finale.

```
        BEGIN
          X := LOWER + DELTA_X * (2.0 * I - 1.0);
          ODD_SUM := ODD_SUM + FX(X)
        END;
      SUM := (7.0 * END_SUM + 14.0 * EVEN_SUM
              + 16.0 * ODD_SUM + END_COR * DELTA_X)
              * DELTA_X/15.0;
      WRITELN(PIECES :5, SUM)
    UNTIL ABS(SUM - SUM1) <= ABS(TOL * SUM)
  END (* simps *);
```

Figura 9.12 — Procedura SIMPS con correzione finale (segue).

Eseguite il programma e confrontate i risultati con la Figura 9.13. Questa tecnica produce la convergenza più rapida rispetto alle altre tecniche usate finora. D'altra parte, come con il metodo trapezoidale, il termine di correzione finale non può essere usato se la tangente è prossima all'infinito. Inoltre, se la funzione ha tangente zero in entrambi gli estremi dell'intervallo, il termine di correzione finale è zero.

Nel prossimo paragrafo studieremo una tecnica di integrazione numerica un po' più complicata, e la implementeremo con un programma Pascal; eseguiremo poi il programma con le stesse tre funzioni a cui abbiamo applicato il metodo di Simpson. Potremo così confrontare i due metodi.

```
   2  2.54815
   4  2.16287
   8  2.19490
  16  2.19713
  32  2.19722
  64  2.19722

area =  2.19722
```

Figura 9.13 — Integrazione con la regola di Simpson con correzione finale.

IL METODO DI ROMBERG

Il metodo di Simpson, che usa un sistema di equazioni del secondo ordine, rappresenta un superamento del metodo trapezoidale che abbiamo usato con equazioni del primo ordine. Possiamo ora tentare di perfezionare ulteriormente il metodo di integrazione numerica sostituendo la curva originale con un insieme di funzioni polinomiali del terzo ordine o di ordine superiore. Esiste un altro approccio, noto come il metodo di integrazione di Romberg, con cui l'area viene calcolata con il metodo trapezoidale, evitando però gli errori tipici di questo metodo con una interpolazione.

Indichiamo la sequenza dei valori del metodo trapezoidale con t_{11}, t_{21}, t_{31}, etc., che vengono assegnati alla prima colonna della matrice bidimensionale T. I valori interpolati di primo livello sono t_{12}, t_{22}, t_{32}, etc., e occupano la seconda colonna della matrice T. Interpolando questi ultimi valori otteniamo una terza colonna t_{13}, t_{23}, t_{33}, etc.

$$\begin{bmatrix} t_{11} & t_{12} & t_{13} & \dots \\ t_{21} & t_{22} & t_{23} & \dots \\ t_{31} & t_{32} & t_{33} & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix}$$

Continuando così troveremo che i valori ottenuti con l'interpolazione convergono rapidamente verso il valore esatto dell'integrale. Il vantaggio di questo metodo è che la funzione deve essere calcolata solo per i valori della prima colonna, che sono quelli della normale regola trapezoidale, e non deve essere calcolata per i valori delle altre colonne. Ogni valore è ottenuto da una combinazione del dato a sinistra e da quello immediatamente sotto questo. Per esempio:

$$t_{12} = \frac{4t_{21} - t_{11}}{3}$$

$$t_{22} = \frac{4t_{31} - t_{21}}{3}$$

$$t_{13} = \frac{16t_{22} - t_{12}}{15}$$

L'algoritmo generale è:

$$T_{ij} = \frac{4^{j-1}t_{i+1,j-1} - t_{i,j-1}}{4^{j-1} - 1}$$

PROGRAMMA PASCAL: INTEGRAZIONE COL METODO DI ROMBERG

La Figura 9.14 mostra un programma che può essere usato per eseguire l'integrazione numerica con il metodo di Romberg. Scrivetelo ed eseguitelo.

```
PROGRAM ROMB1(OUTPUT);

(* integrazione per mezzo del metodo di Romberg *)
(* 24 Dic 80 *)

CONST
   TOL = 1.0E-5;

VAR
   DONE : BOOLEAN;
   SUM, UPPER, LOWER : REAL;

FUNCTION FX(X : REAL) : REAL;
(* dichiara f(x) = 1/x *)
(* si badi ad x = 0 *)

BEGIN
   FX := 1.0/X
END;

PROCEDURE ROMB(LOWER, UPPER, TOL : REAL;
                              VAR ANS : REAL);

(* integrazione numerica con il metodo di Romberg *)

VAR
   NX : ARRAY[1..16] OF INTEGER;
   T : ARRAY[1..136] OF REAL;
   DONE, ERROR : BOOLEAN;
   PIECES, NT, I, II, N, NN,
   L, NTRA, K, M, J : INTEGER;
   DELTA_X, C, SUM, FOTOM, X : REAL;
```

Figura 9.14 — Il metodo di Romberg.

```
BEGIN
  DONE := FALSE;
  ERROR := FALSE;
  PIECES := 1;
  NX[1] := 1;
  DELTA_X := (UPPER - LOWER)/PIECES;
  C := (FX(LOWER) + FX(UPPER)) * 0.5;
  T[1] := DELTA_X * C;
  N := 1;
  NN := 2;
  SUM := C;
  REPEAT
    N := N+1;
    FOTOM := 4.0;
    NX[N] := NN;
    PIECES := PIECES * 2;
    L := PIECES - 1;
    DELTA_X := (UPPER - LOWER)/PIECES;
    (calcola la somma trapezoidale per 2↑(n - 1) + 1 punti *)
    FOR II := 1 TO (L + 1) DIV 2 DO
      BEGIN
        I := II * 2 - 1;
        X := LOWER + I * DELTA_X;
        SUM := SUM + FX(X)
      END;

    T[NN] := DELTA_X * SUM;
    WRITE(PIECES:5, T[NN]);
    NTRA := NX[N-1];
    K := N-1;
    (* calcola la n-sima riga della matrice T *)
    FOR M := 1 TO K DO
```

Figura 9.14 Il metodo di Romberg (segue).

```
        BEGIN
           J := NN + M;
           NT := NX[N-1] + M - 1;
           T[J] := (FOTOM * T[J-1] - T[NT])/(FOTOM - 1.0);
           FOTOM := FOTOM * 4.0
        END;
     WRITELN(J :4, T[J]);
     IF N > 4 THEN
        BEGIN
           IF T[NN+1] <> 0.0 THEN
              IF (ABS(T[NTRA+1] - T[NN+1]) <= ABS(T[NN+1] * TOL))
                 OR (ABS(T[NN-1] - T[J]) <= ABS(T[J] * TOL)) THEN
                       DONE := TRUE
              ELSE IF N > 15 THEN
                 BEGIN
                    DONE := TRUE;
                    ERROR := TRUE
                 END
        END; (* IF n > 4 *)
     NN := J + 1
   UNTIL DONE;
   ANS := T[J]
END (* romberg *);

BEGIN (* programma principale *)
   LOWER := 1.0;
   UPPER := 9.0;
   WRITELN;
   ROMB(LOWER, UPPER, TOL; SUM);
   WRITELN;
   WRITELN(' area = ', SUM)
END.
```

Figura 9.14 Il metodo di Romberg (segue).

Prima esecuzione del programma di Romberg

I valori ottenuti con la regola trapezoidale sono stampati ad ogni ciclo, come prima; inoltre sono stampati i valori ottenuti tramite interpolazione nella parte destra della matrice. I risultati si presentano come in Figura 9.15.

```
 2 3.02222   3 2.54815
 4 2.46349   6 2.25919
 8 2.27341  10 2.20472
16 2.21733  15 2.19773
32 2.20234  21 2.19724
64 2.19851  28 2.19723

area =  2.19723
```

Figura 9.15 — Integrazione con il metodo di Romberg.

Confrontando i valori della Figura 9.15 con quelli delle tabelle precedenti si vede che il metodo di Romberg converge anche più rapidamente (in questo caso) del metodo di Simpson.

Seconda esecuzione del metodo di Romberg
Una funzione esponenziale

Modificazione il metodo di Romberg per calcolare l'integrale

$$\int_0^1 e^{-x^2}\,dx$$

Modificate la funzione FX come segue

```
FUNCTION FX(X : REAL) : REAL;
BEGIN
  FX : = EXP(-X* X)
END;
```

e i limiti definiti nelle prime linee del programma principale:

```
LOWER : = 0.0;
UPPER : = 1.0;
```

Eseguite il metodo di Romberg con la nuova formula; i risultati si presenteranno come in Figura 9.16. Notate che in questo caso la convergenza non è altrettanto rapida che con il metodo di Simpson.

```
 2  0.731370    3  0.747180
 4  0.742984    6  0.746834
 8  0.745866   10  0.746824
16  0.746585   15  0.746824

area =  0.746824
```

Figura 9.16 — Integrazione di e^{-x^2} con il metodo di Romberg.

Terza esecuzione del programma di Romberg
Una funzione periodica

Torniamo all'integrale

$$\int_0^{4\pi} \sin^2 x \, dx$$

Se tentiamo di risolvere questa equazione con il metodo di Romberg, otterremo un risultato nullo per le prime due approssimazioni. Se però abbiamo l'accortezza di usare una tolleranza relativa invece che assoluta troveremo la soluzione corretta dopo un certo numero di iterazioni.

Fate una copia del programma di Romberg, e fate le seguenti modifiche. Aggiungete l'istruzione

```
PI = 3.141593;
```

nella sezione CONST all'inizio del programma; modificate la funzione FX

```
BEGIN
  FX : = SQR(SIN(X))
END;
```

e la prima e la seconda linea del programma principale:

```
LOWER : = 0.0;
UPPER : = 4.0 * PI;
```

Eseguite il programma e confrontate i risultati con la Figura 9.17.

Infine, nel prossimo paragrafo amplieremo il programma di integrazione di Romberg per trattare una funzione che tende all'infinito in uno degli estremi dell'intervallo di integrazione. Per risolvere questo problema useremo una tecnica di aggiustamento dell'ampiezza delle sezioni man mano che si avvicinano a questo estremo fino ad arrivare a un valore sufficientemente preciso dell'area.

FUNZIONI CHE TENDONO ALL'INFINITO AD UN ESTREMO DELL'INTERVALLO DI INTEGRAZIONE

In tutti i metodi usati finora abbiamo usato sezioni di ampiezza uguale in tutto l'intervallo. È evidente che la convergenza è più difficile, ed è quindi necessaria una suddivisione più fitta, quando la funzione ha una tangente molto grande, mentre è sufficiente una suddivisione meno fitta quando la tangente è piccola. Inoltre una funzione può essere infinita ad un estremo pur avendo area finita in quell'intervallo.

Consideriamo per esempio l'integrale dell'inverso della radice quadrata di x nell'intervallo 0, 1:

$$\int_0^1 \frac{dx}{\sqrt{x}}$$

Il valore esatto dell'integrale, calcolato con integrazione diretta, è 2.0.

Se tentiamo di calcolare questo integrale con uno dei programmi precedenti, avremo subito un problema: il valore di f(a) all'estremo sinistro dell'intervallo di integrazione è infinito, e quindi dobbiamo scegliere un altro estremo inferiore per l'intervallo. Se scegliamo 10^{-11} ci vorrà molto tempo per ottenere la convergenza, ma se scegliamo un valore più grande, come 0.01, il risultato non sarà abbastanza preciso.

PROGRAMMA PASCAL: SEZIONI DI AMPIEZZA VARIABILE PER UNA FUNZIONE INFINITA

Un metodo per risolvere questo problema consiste nell'iniziare l'integrazione da un valore un po' più grande di zero, ad esempio 0.1. L'integrale viene quindi calcolato tra 0.1 e 1.0.

```
 2 9.64326e-12    3 8.57179e-12
 4 8.08070e-12    6 7.49238e-12
 8 6.28319       10 9.07793
16 6.28319       15 6.08755
32 6.28319       21 6.28633

area =  6.28633
```

Figura 9.17 — Integrazione di $\sin^2 x$ con il metodo di Romberg.

Per verificare la bontà di questa scelta, calcoliamo poi l'area di una regione a sinistra, tra 0.01 e 0.1; se questa è significativa dobbiamo aggiungerla alla precedente. Poi calcoliamo l'area della regione compresa tra 0.001 e 0.01, e così via, avvicinandoci sempre più allo zero, e valutando ogni volta il valore della nuova area. Il programma della Figura 9.18 usa questo approccio con il metodo di Romberg.

Esecuzione del programma

Scrivete il programma ed eseguitelo; è simile al precedente e quindi vi conviene modificare una copia della Figura 9.14.

```
PROGRAM ROMB3(OUTPUT);
(* integrazione per mezzo del metodo di Romberg *)
(* 9 Gen 81 *)

CONST
  TOL = 1.0E-5;

VAR
  DONE : BOOLEAN;
  SUMT, SUM, UPPER, LOWER : REAL;
```

Figura 9.18 — Integrazione con il metodo di Romberg con sezioni di ampiezza variabile.

```
FUNCTION FX(X : REAL) : REAL;

(* dichiara f(x) = 1/sqrt(x) *)
(* si badi ad x = 0 *)

BEGIN
   FX := 1.0/SQRT(X)
END;

PROCEDURE ROMB(LOWER, UPPER, TOL : REAL;
                                VAR ANS : REAL);

(* integrazione numerica con metodo di Romberg *)

VAR
   NX : ARRAY[1..16] OF INTEGER;
   T : ARRAY[1..136] OF REAL;
   DONE, ERROR : BOOLEAN;
   PIECES, NT,I, II, N, NN,
   L, NTRA, K, M, J : INTEGER;
   DELTA_X, C, SUM, FOTOM, X : REAL;

BEGIN
   DONE := FALSE;
   ERROR := FALSE;
   PIECES := 1;
   NX[1] := 1;
   DELTA_X := (UPPER - LOWER)/PIECES;
   C := (FX(LOWER) + FX(UPPER)) * 0.5;
   T[1] := DELTA_X * C;
   N := 1;
   NN := 2;
   SUM := C;
```

Figura 9.18 — Integrazione con il metodo di Romberg con sezioni di ampiezza variabile (segue).

```
REPEAT
   N := N + 1;
   FOTOM := 4.0;
   NX[N] := NN;
   PIECES := PIECES * 2;
   L := PIECES - 1;
   DELTA_X := (UPPER - LOWER)/PIECES;
   (* calcola la somma trapezoidale per 2↑(n - 1) + 1 punti *)
   FOR II := 1 TO (L + 1) DIV 2 DO

    BEGIN
       I := II * 2 - 1;
       X := LOWER + I * DELTA_X;
       SUM := SUM + FX(X)
    END;
T[NN] := DELTA_X * SUM;
NTRA := NX[N-1];
K := N - 1;
(* calcola la n-sima riga della matrice T *)
FOR M := 1 TO K DO
   BEGIN
      J := NN + M;
      NT := NX[N-1] + M - 1;
      T[J] := (FOTOM*T[J-1] - T[NT])/(FOTOM - 1.0);
      FOTOM := FOTOM * 4.0
   END;
IF N > 4 THEN
   BEGIN
      IF T[NN+1] <> 0.0 THEN
         IF (ABS(T[NTRA+1] - T[NN+1]) <= ABS(T[NN+1] * TOL))
            OR (ABS(T[NN-1] - T[J]) <= ABS(T[J] * TOL)) THEN
```

Figura 9.18 — Integrazione con il metodo di Romberg con sezioni di ampiezza variabile (segue).

```
                DONE := TRUE
            ELSE IF N > 15 THEN
              BEGIN
                DONE := TRUE;
                ERROR := TRUE
              END
        END; (* IF n > 4 *)
      NN := J + 1
    UNTIL DONE;
    ANS := T[J]
  END (* romberg *);


  BEGIN (* programma principale *)
    LOWER := 0.1;
    UPPER := 1.0;
    WRITELN;
    SUMT := 0.0;
    WRITELN(' nuova area totale estremi',
            'inferiore superiore');


    REPEAT
      ROMB(LOWER, UPPER, TOL, SUM);
      UPPER := LOWER;
      LOWER := 0.1 * UPPER;
      SUMT := SUMT + SUM;
      WRITELN(SUM :9:6,' ', SUMT :9:5,
            ' ',LOWER, ' ', UPPER)
    UNTIL ABS(SUM) < TOL
  END.
```

Figura 9.18 — Integrazione con il metodo di Romberg con sezioni di ampiezza variabile (segue).

L'istruzione WRITELN della procedura ROMB è stata eliminata, in modo da ottenere solo la stampa del valore finale di ogni area. Il programma principale chiama ripetutamente la procedura ROMB fornendo limiti sempre più vicini allo zero. Quando la nuova area è minore della tolleranza il procedimento ha termine. I risultati si presentano come in Figura 9.19.

nuova area	area totale	estremi inferiore	superiore
1.36754	1.36754	1.00000e-2	1.00000e-1
0.432456	1.80000	1.00000e-3	1.00000e-2
0.136754	1.93675	1.00000e-4	1.00000e-3
0.043246	1.98000	1.00000e-5	1.00000e-4
0.013675	1.99368	1.00000e-6	1.00000e-5
0.004325	1.99800	1.00000e-7	1.00000e-6
0.001368	1.99937	1.00000e-8	1.00000e-7
0.000432	1.99980	1.00000e-9	1.00000e-8
0.000137	1.99994	1.00000e-10	1.00000e-9
0.000043	1.99998	1.00000e-11	1.00000e-10
0.000014	1.99999	1.00000e-12	1.00000e-11
0.000004	2.00000	1.00000e-13	1.00000e-12

Figura 9.19 — Integrazione di $1/\sqrt{x}$ vicino allo zero.

Se non si eliminano le stampe intermedie nella procedura ROMB, si può vedere che ogni regione è suddivisa in 64 sezioni; l'ampiezza di ogni regione è un decimo di quella immediatamente a destra, e l'estremo sinistro finale è 10^{-13}. Se si calcolasse l'integrale tra 10^{-13} e 1.0 in una volta sola, sarebbe necessario più di un milione di milioni di sezioni di uguale ampiezza, e questo richiederebbe un tempo enorme per ottenere il risultato esatto.

SOMMARIO

In questo capitolo abbiamo sviluppato e confrontato tre diversi programmi di integrazione numerica. I primi due — la regola trapezoidale e il metodo di Simpson — usano la tecnica di correzione finale per rendere più preciso il valore dell'ultima approssimazione dell'area sottesa dalla curva. Il terzo metodo, più sofisticato — il metodo di Romberg — usa una matrice di interpolazioni progressive, e non richiede la correzione dell'errore. Abbiamo provato questi metodi su diverse funzioni, e abbiamo usato una versione perfezionata del metodo di Romberg per calcolare l'area sottesa dalla curva di una funzione infinita.

Tutte le integrazioni, tranne l'ultima, erano eseguite su funzioni analitiche con suddivisioni di uguale ampiezza. Tecniche diverse sono necessarie per insiemi discreti di dati; un approccio è quello di interpolare i dati con una funzione polinomiale, usando una delle tecniche viste nei capitoli precedenti, e integrare poi questa funzione.

CAPITOLO 10

EQUAZIONI NON LINEARI DI INTERPOLAZIONE DI CURVE

INTRODUZIONE

Nei Capitoli 5 e 7 abbiamo sviluppato programmi per determinare i coefficienti di diverse equazioni di interpolazione di curve. Avendo scelto funzioni di approssimazione con coefficienti lineari, le equazioni risultanti erano lineari, e quindi facilmente risolte. Può però essere necessario, talvolta, scegliere funzioni di approssimazione con coefficienti non lineari, e in questo caso il calcolo dei coefficienti può essere notevolmente più complesso.

In questo capitolo studieremo due diverse tecniche di interpolazione non lineare di curve. In uno di questi metodi, renderemo dapprima lineare la funzione di approssimazione e troveremo la soluzione nella forma lineare; con questo metodo eseguiremo l'interpolazione di due insiemi di dati. Prima eseguiremo l'interpolazione della forma linearizzata di quella che si chiama funzione razionale, per un insieme di dati che rappresentano il fattore di Clausing. Poi interpoleremo una funzione esponenziale razionalizzata per un insieme di dati che rappresenta la diffusione dello zinco nel rame in un certo intervallo di temperature.

Il secondo approccio all'interpolazione non lineare sarà più diretto. Vedremo che non esiste una tecnica generale per trattare funzioni di approssimazione con coefficienti non lineari, e cercheremo un metodo specifico per equazioni esponenziali. Questo metodo comporta l'eliminazione di uno dei coefficienti e la risoluzione delle equazioni risultanti col metodo di Newton, che abbiamo studiato nel Capitolo 8.

Cominciamo con il primo esempio del metodo di linearizzazione.

LINEARIZZAZIONE DELLA FUNZIONE RAZIONALE

Una funzione di approssimazione non lineare, comunemente usata, è la funzione razionale, ottenuta dal rapporto tra due polinomi:

$$y = \frac{A_1 + A_3x + A_5x^2 \ldots}{1 + A_2x + A_4x^2 + A_6x^3 \ldots}$$

In questa espressione x è la variabile indipendente, y la variabile dipendente, e A_1, A_2, etc. sono i coefficienti.

La funzione razionale non è lineare, ma può essere linearizzata con le seguenti operazioni: si moltiplicano entrambi i membri delle equazioni per il denominatore, e si ottiene:

$$y(1 + A_2x + A_4x^2 + A_6x^3 \ldots) = A_1 + A_3x + A_5x^2 \ldots$$

I termini dell'equazione precedente possono essere manipolati in modo da ottenere:

$$y = A_1 - A_2xy + A_3x - A_4x^2y + A_5x^2 - A_6x^3y \ldots$$

Alcuni dei termini a destra contengono la variabile dipendente y insieme a quella indipendente x. Ricordiamo che sia x che y sono matrici di valori noti, mentre le incognite da determinare sono i coefficienti A_1, A_2, ...A_n. Tutti questi coefficienti sono lineari, e quindi possono essere determinati con i metodi visti nei Capitoli 5 e 7.

PROGRAMMA PASCAL: IL FATTORE DI CLAUSING INTERPOLATO CON LA FUNZIONE RAZIONALE

La Figura 10.1 contiene un programma per l'interpolazione col metodo dei minimi quadrati della forma linearizzata della funzione razionale. Il programma si può ricavare dalla Figura 7.3 del Capitolo 7.

I dati di questo programma rappresentano il fattore di Clausing come funzione del rapporto lunghezza/raggio (L/r) per fori cilindrici. Quando delle molecole con grande "libero cammino medio" si diffondo attraverso un foro cilindrico, alcune di esse urtano le pareti del foro e rimbalzano nella direzione opposta; le altre proseguono fino all'altro capo del foro. Il fattore di Clausing dà la frazione delle molecole che entrando da un lato del foro cilindrico escono dall'altro lato, e può avere un valore compreso tra 0 e 1; naturalmente questo valore diminuisce all'aumentare della lunghezza del foro e al diminuire del raggio, perciò nella formula di calcolo diretto del fattore di Clausing compare il rapporto L/r.

```
PROGRAM FITPOL(OUTPUT);
  (* 30 Dic 80 *)

(* Programma pascal che esegue una *)
(* interpolazione lineare coi minimi quadrati *)
(* al rapporto di 2 polinomi *)
(* È necessaria la procedura separata GAUSSJ *)

CONST
  MAXR = 20; (* punti *)
  MAXC = 4; (* termini polinomiali *)

TYPE
  ARY   = ARRAY[1..MAXR] OF REAL;
  ARYS  = ARRAY[1..MAXC] OF REAL;
  ARY2  = ARRAY[1..MAXR, 1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXC, 1..MAXC] OF REAL;

VAR
  X, Y, Y_CALC,
  RESID          : ARY;
  COEF, SIG      : ARYS;
  NROW, NCOL     : INTEGER;
  CORREL_COEF    : REAL;

PROCEDURE GET_DATA
          (VAR X    : ARY; (* variabile indipendente *)
           VAR Y    : ARY; (* variabile dipendente *)
           VAR NROW : INTEGER); (* lunghezza dei vettori *)

VAR
  I : INTEGER;
```

Figura 10.1 — Il fattore di Clausing interpolato con il rapporto tra due polinomi.

```
BEGIN
(* Fattori di Clausing *)
   NROW := 10;
   X[1] := 0.1; Y[1] := 0.9524;
   X[2] := 0.2; Y[2] := 0.9092;
   X[3] := 0.5; Y[3] := 0.8013;
   X[4] := 1.0; Y[4] := 0.6720;
   X[5] := 1.2; Y[5] := 0.6322;
   X[6] := 1.5; Y[6] := 0.5815;
   X[7] := 2.0; Y[7] := 0.5142;
   X[8] := 3.0; Y[8] := 0.4201;
   X[9] := 4.0; Y[9] := 0.3566;
   X[10] := 6.0; Y[10] := 0.2755
END (* procedure get_data *);

PROCEDURE WRITE_DATA;
(* stampa i risultati *)

VAR
   I: INTEGER;

BEGIN
   WRITELN;
   WRITELN;
   WRITELN('   I    X    Y    Y CALC    RESID');

   FOR I := 1 TO NROW DO
      WRITELN (I: 3, X[I]: 8: 1, Y[I]: 9: 4,
         Y_CALC[I]: 9: 4, RESID[I]: 9: 4);
   WRITELN;
   WRITELN('coefficienti errori');
   WRITELN(COEF[1]:8:5, '  ', SIG[1], 'Termine costante');
```

Figura 10.1 — Il fattore di Clausing interpolato con il rapporto tra due polinomi (segue).

```
   FOR I := 2 TO NCOL DO
      WRITELN
         (COEF[I]:8:5, '  ', SIG[I]); (* altri termini *)
   WRITELN;
   WRITELN('Il coefficiente di correlazione è', CORREL_COEF: 8: 5)
END (* write_data *);

(* PROCEDURE square(x : ary2;
                        y : ary;
                    VAR a : ary2s;
                    VAR g : arys;
                 nrow,ncol : integer);
extern; *)
(*$F SQUARE.PAS *)

(* PROCEDURE gaussj
   (VAR b      : ary2s;
        y      : arys;
    VAR coef   : arys;
        ncol   : integer;
    VAR error  : boolean);
extern; *)
(*$F GAUSSJ.PAS *)

PROCEDURE LINFIT(X,          (* variabile indipendente *)
                     Y : ARY; (* variabile dipendente *)
        VAR Y_CALC: ARY; (* var. dipendente calcolata *)
        VAR RESID   : ARY; (* matrice dei testi *)

        VAR  COEF   : ARYS; (* coefficienti *)
        VAR  SIG    : ARYS; (* errori nei coefficienti *)
             NROW   : INTEGER; (* lunghezza delle matrici *)
```

Figura 10.1 — Il fattore di Clausing interpolato con il rapporto tra due polinomi (segue).

```
        VAR  NCOL   : INTEGER); (* numero di termini *)

(* interpolazione coi minimi quadrati di *)
(* nrow coppie di punti x e y *)
(* Sono necessarie le procedure separate:
   SQUARE — formazione della matrice dei coefficienti
   GAUSSJ — eliminazione di Gauss-Jordan *)

VAR
   XMATR  : ARY2; (* matrice dei dati *)
   A      : ARY2S; (* matrice dei coefficienti *)
   G      : ARYS; (* vettore delle costanti *)
   ERROR  : BOOLEAN;
   I, J, NM : INTEGER;
   XI, YI, YC, SRS, SEE,
   SUM_Y, SUM_Y2: REAL;

BEGIN (* procedure linfit *)
   NCOL := 4 (* numero di termini *)
   FOR I := 1 TO NROW DO
      BEGIN (* carica la matrice X *)
         XI := X[I];
         YI := Y[I];
         XMATR[I, 1] := 1.0 (* prima colonna *);
         XMATR[I, 2] := —XI * YI;
         XMATR[I, 3] := XI;
         XMATR[I, 4] := —SQR(XI) * YI
      END;

SQUARE(XMATR, Y, A, G, NROW, NCOL);
GAUSSJ(A, G, COEF, NCOL, ERROR);
```

Figura 10.1 — Il fattore di Clausing interpolato con il rapporto tra due polinomi (segue).

```
  SUM_Y := 0.0;
  SUM_Y2 := 0.0;
  SRS := 0.0;
  FOR I := 1 TO NROW DO
    BEGIN
      XI := X[I];
      YI := Y[I];
      YC := COEF[1]
        +(-COEF[2] * YI + COEF[3] - COEF[4] * XI * YI) * XI;
      Y_CALC[I] := YC;
      RESID[I] := YC - YI;
      SRS := SRS + SQR(RESID[I]);
      SUM_Y := SUM_Y + YI;
      SUM_Y2 := SUM_Y2 + YI * YI

    END;
  CORREL_COEF := SQRT(1.0 - SRS /
          (SUM_Y2 - SQR(SUM_Y)/NROW));
  IF NROW = NCOL THEN NM := 1
  ELSE NM := NROW - NCOL;
  SEE := SQRT(SRS/NM);
  FOR I := 1 TO NCOL DO (* errori nella soluzione *)
    SIG[I] := SEE * SQRT(A[I, I])
END (* linfit *);


BEGIN (* programma principale *)
  GET_DATA(X, Y, NROW);
  LINFIT(X, Y, Y_CALC, RESID, COEF, SIG, NROW, NCOL);
  WRITE_DATA
END.
```

Figura 10.1 — Il fattore di Clausing interpolato con il rapporto tra due polinomi (segue).

Esecuzione del programma

Scrivete il programma ed eseguitelo. La matrice viene calcolata per quattro termini, corrispondenti a un numeratore del primo ordine e a un denominatore del secondo. Termini addizionali possono essere facilmente aggiunti alla funzione di approssimazione. Le variabili MAXC e NCOL devono essere modificate per contenere il numero effettivo dei termini; inoltre devono essere aggiunte le espressioni

```
MATR [I,5]: = MATR [I,3] * XI;
MATR [I,6]: = MATR [I,4] * XI;
```

nella procedura LINFIT, se si vogliono aggiungere termini addizionali. I risultati mostrati in Figura 10.2 corrispondono all'equazione:

$$y = \frac{1.0017 + 0.237x}{1 + 0.7522x + 0.0912x^2}$$

Se i dati sono interpolati con una funzione polinomiale regolare invece che con la funzione razionale, l'interpolazione non sarà altrettanto buona (a parità del numero di coefficienti nella funzione di approssimazione).

Nel prossimo esempio del metodo di linearizzazione, studieremo un'equazione esponenziale. Più avanti, in questo capitolo, interpoleremo la stessa equazione con un altro approccio più diretto.

I	X	Y	Y CALC	RESID
1	0.1	0.9524	0.9529	0.0005
2	0.2	0.9092	0.9090	-0.0002
3	0.5	0.8013	0.8006	-0.0007
4	1.0	0.6720	0.6719	-0.0001
5	1.2	0.6322	0.6324	0.0002
6	1.5	0.5815	0.5818	0.0003
7	2.0	0.5142	0.5146	0.0004
8	3.0	0.4201	0.4199	-0.0002
9	4.0	0.3566	0.3563	-0.0003
10	6.0	0.2755	0.2756	0.0001

coefficienti	errori	
1.00169	4.60364e-4	Termine costante
0.75220	1.37135e-2	
0.23704	1.22040e-2	
0.09124	5.14729e-3	

Il coefficiente di correlazione è 1.00000

Figura 10.2 — Il fattore di Clausing in funzione di L/r interpolato con una funzione razionale.

LINEARIZZAZIONE DELL'EQUAZIONE ESPONENZIALE

Una delle più comuni equazioni non lineari ha la forma:

$$y = Ae^{Bx}$$

dove x è la variabile indipendente, y la variabile dipendente, A e B i coefficienti; questa equazione si incontra frequentemente in problemi scientifici e tecnici in quanto rappresenta la soluzione di un'equazione differenziale del primo ordine.

L'equazione precedente può essere linearizzata calcolando il logaritmo di entrambi i membri; si ottiene

$$\ln y = \ln A + Bx$$

In questa forma, la variabile dipendente è ln y, e i coefficienti da determinare di ventano ln A e B; poichè sono lineari, possiamo ottenere un'interpolazione con il metodo dei minimi quadrati con la procedura LINFIT del Capitolo 5.

PROGRAMMA PASCAL: UN'INTERPOLAZIONE CON LA CURVA ESPONENZIALE PER LA DIFFUSIONE DELLO ZINCO NEL RAME

In Figura 10.3 c'è un programma per l'interpolazione con i minimi quadrati con l'equazione esponenziale linearizzata; il programma si può ottenere dalla Figura 7.3 del Capitolo 7. Il valore di NCOL è diventato 2, ed è stato eliminato il calcolo dell'errore standard. I dati della procedura GET_DATA rappresentano la diffusione dello zinco nel rame in un intervallo di temperatura da 600°C a 900°C. L'equazione della diffusione è:

$$D = D_0 e^{-Q/RT}$$

dove D è il coefficiente di diffusione in cm^2/sec, D_0 è la costante di diffusione nella stessa unità di misura, Q l'energia di attivazione in cal/mole, R la costante dei gas e T la temperatura in gradi Kelvin. La variabile indipendente x è l'inverso della temperatura in gradi Kelvin; la variabile dipendente y è il logaritmo del coefficiente di diffusione.

```
PROGRAM DIFFUS(OUTPUT);
  (* 30 Dic 80 *)
(* Programma pascal che esegue una *)
(* interpolazione coi minimi quadrati *)
(* per la diffusione dello Zn nel Cu *)
(* È necessaria la procedura esterna GAUSSJ *)
CONST
  MAXR = 20 (* punti *);
  MAXC = 4 (* termini polinomiali *);
  R    = 1.987 (* costante del gas *);
TYPE
  ARY   = ARRAY[1..MAXR] OF REAL;
  ARYS  = ARRAY[1..MAXC] OF REAL;
  ARY2  = ARRAY[1..MAXR, 1..MAXC] OF REAL;
  ARY2S = ARRAY[1..MAXC, 1..MAXC] OF REAL;
VAR
  X, Y, Y_CALC, T, D,
  RESID          : ARY;
  COEF, SIG      : ARYS;
  NROW, NCOL     : INTEGER;
  CORREL_COEF, SRS : REAL;
PROCEDURE GET_DATA(VAR X, Y, T, D : ARY;
                   VAR NROW      : INTEGER);
(* riceve i valori per nrow e le matrici t, d *)
VAR
  I : INTEGER;
BEGIN
  NROW := 7;
  T[1] := 600.0; D[1] := 1.4E-12;
  T[2] := 650.0; D[2] := 5.5E-12;
  T[3] := 700.0; D[3] := 1.8E-11;
  T[4] := 750.0; D[4] := 6.1E-11;
  T[5] := 800.0; D[5] := 1.6E-10;
```

Figura 10.3 — Un'interpolazione con i minimi quadrati per un'equazione esponenziale linearizzata.

```
    T[6] := 850.0; D[6] := 4.4E-10;
    T[7] := 900.0; D[7] := 1.2E-9;
    FOR I := 1 TO NROW DO
      BEGIN
        X[I] := 1.0/(T[I] + 273.0);
        Y[I] := LN(D[I])
      END
  END (* procedure get_data *);

  PROCEDURE WRITE_DATA;
  (* stampa i risultati *)

  VAR
    I: INTEGER;

  BEGIN
    WRITELN;
    WRITELN;
    WRITELN('   I    T C    D    ',
            '    D Calc');
    FOR I := 1 TO NROW DO
      WRITELN(I:3, T[I]:8:0, D[I], ' ',
      Y_CALC[I]);
    WRITELN;
    WRITELN('coefficienti');
    WRITELN(COEF[1], '      Termine costante');
    FOR I := 2 TO NCOL DO
      WRITELN
      (COEF[I]); (* altri termini *)
    WRITELN;
    WRITELN(' D0 = ', (EXP(COEF[1])) :7:2,' cm sq/sec.');
    WRITELN(' Q = ', (-R * COEF[2]/1000.0) :8:2,
              ' kcal/mole');
    WRITELN; WRITELN(' SRS = ', SRS :7:3)
  END (* write_data *);
```

Figura 10.3 — Un'interpolazione con i minimi quadrati per un'equazione esponenziale linearizzata (segue).

```
(* PROCEDURE square(x : ary2;
                     y: ary;
                 VAR a : ary2s;
                 VAR g : arys;
            nrow,ncol : integer);
extern; *)
(*$F SQUARE.PAS *)

(* PROCEDURE gaussj
   (VAR b     : ary2s;
        y     : arys;
    VAR coef  : arys;
        ncol  : integer;
    VAR error : boolean);
extern; *)
(*$F GAUSSJ.PAS *)

PROCEDURE LINFIT(X,          (* variabile indipendente *)
                 Y : ARY; (* variabile dipendente *)
      VAR Y_CALC: ARY; (* var. dipendente calcolata *)
      VAR RESID   : ARY; (* matrice dei resti *)
      VAR  COEF   : ARYS; (* coefficienti *)
      VAR  SIG    : ARYS; (* errori nei coefficienti *)
           NROW   : INTEGER; (* lunghezza delle matrici *)
      VAR  NCOL   : INTEGER); (* numero di termini *)

(* interpolazione coi minimi quadrati di *)
(* nrow coppie di punti x, y *)
(* Sono necessarie le procedure separate:
   SQUARE — formazione della matrice dei coefficienti
   GAUSSJ — eliminazione di Gauss-Jordan *)
```

Figura 10.3 — Un'interpolazione con i minimi quadrati per un'equazione esponenziale linearizzata (segue).

```
VAR
   XMATR  : ARY2 (* matrice dei dati *);
   A      : ARY2S (* matrice dei coefficienti *);
   G      : ARYS (* vettore delle costanti *);
   ERROR  : BOOLEAN;
   I, J, NM: INTEGER;
   SEE, A1 : REAL;

BEGIN (* procedure linfit *)
   NCOL := 2 (* numero di termini *);
   FOR I := 1 TO NROW DO
      BEGIN (* carica la matrice X *)
         XMATR[I, 1] := 1.0 (* prima colonna *)
         XMATR[I, 2] := X[I] (* seconda colonna *)
      END;
   SQUARE(XMATR, Y, A, G, NROW, NCOL);
   GAUSSJ(A, G, COEF, NCOL, ERROR);
   SRS := 0.0;
   A1 := EXP(COEF[1]);
   FOR I := 1 TO NROW DO
      BEGIN
         Y_CALC[I] := A1 * EXP(COEF[2] * X[I]);
         IF Y[I] <> 0.0 THEN RESID[I] := Y_CALC[I]/Y[I] - 1.0
         ELSE RESID[I] := Y[I]/Y_CALC[I] - 1.0;
         SRS := SRS + SQR(RESID[I])
      END
END (* linfit *);

BEGIN (* programma principale *)
   GET_DATA(X, Y, T, D, NROW);
   LINFIT(X, Y, Y_CALC, RESID, COEF, SIG, NROW, NCOL);
   WRITE_DATA
END.
```

Figura 10.3 — Un'interpolazione con i minimi quadrati per un'equazione esponenziale linearizzata (segue).

Esecuzione del programma

Compilate il programma, eseguitelo e confrontate i risultati con la Figura 10.4. La costante di diffusione (D0) è l'esponenziale del primo coefficiente:

```
D0 : = EXP(COEF [1])
```

L'energia di attivazione Q si ottiene dal prodotto del secondo coefficiente (COEF [2]) per la costante dei gas, con il segno invertito:

```
Q : = -R*COEF[2]
```

In questo esempio, viene stampata la somma dei quadrati dei residui, SRS, invece dell'errore standard sui coefficienti, in quanto la trasformazione non lineare della funzione di approssimazione rende non significativi questi "sigma". Il calcolo di questa forma di SRS è discusso più esaurientemente nel prossimo paragrafo, dove svilupperemo un secondo approccio di interpolazione non lineare.

```
 I     T C        D            D Calc
 1     600.  1.40000e-12    1.31283e-12
 2     650.  5.50000e-12    5.43316e-12
 3     700.  1.80000e-11    1.94311e-11
 4     750.  6.10000e-11    6.13542e-11
 5     800.  1.60000e-10    1.74041e-10
 6     850.  4.40000e-10    4.49924e-10
 7     900.  1.20000e-9     1.07266e-9

 Coefficienti
-1.13952          Termine costante
-2.28895e4

 D0 =     0.32 cm sq/sec.
 Q  =    45.48 kcal/mole

 SRS =  7.000
```

Figura 10.4 — La diffusione dello zinco nel rame (interpolazione lineare).

SOLUZIONE DIRETTA DELL'EQUAZIONE ESPONENZIALE

Nel paragrafo precedente, l'equazione esponenziale

$$y = Ae^{Bx}$$

è stata linearizzata con il logaritmo:

$$\ln y = \ln A + Bx$$

Poi è stata fatta un'interpolazione con i minimi quadrati di questa forma linearizzata. Ma i coefficienti che forniscono la minima somma dei quadrati dei residui (SRS) per la forma linearizzata, non la forniscono per l'equazione originale non lineare. Dobbiamo quindi considerare una soluzione diretta, con i minimi quadrati, dell'equazione non lineare.

In questo paragrafo ricaveremo le equazioni di interpolazione relative all'equazione originale non linearizzata. Se affrontiamo la soluzione dell'equazione non lineare nello stesso modo di quella lineare, otterremo il valore di SRS per la funzione di approssimazione. Mettendo poi a zero le derivate di SRS rispetto ad ogni coefficiente, si ottiene un'equazione per ogni incognita, che però, essendo non lineare, non può in genere essere risolta.

Non esiste un approccio universale per la soluzione di equazioni non lineari, ma ci sono diverse tecniche per i diversi casi. Nel caso della funzione esponenziale, la soluzione è relativamente facile. Seguiremo il solito algoritmo di interpolazione di una curva fino al calcolo delle derivate di SRS. Poi studieremo il modo di eliminare uno dei coefficienti dall'equazione, e risolveremo la funzione risultante con il metodo di Newton.

Calcolo di SRS

I residui per l'equazione

$$y = Ae^{Bx}$$

possono essere definiti come

$$r = Ae^{Bx} - y$$

D'altra parte, se i dati sono tutti calcolati con lo stesso grado di precisione relativa, indipendentemente dalla loro grandezza, è più significativo usare un residuo relativo

$$r = (Ae^{Bx} - y) / y$$

o

$$r = (Ae^{Bx} / y) - 1$$

I residui in questa nuova forma vengono elevati al quadrato e poi sommati, per ottenere SRS. Vengono poi calcolate le derivate di SRS rispetto ad A e B, e le equazioni risultanti sono messe a zero. Questo approccio è lo stesso di prima; ci sono due equazioni e due incognite.

$$A\Sigma(e^{2Bx} / y^2) - \Sigma(e^{Bx} / y) = 0$$

e

$$A\Sigma(xe^{2Bx} / y^2) - \Sigma(xe^{Bx} / y) = 0$$

Questa volta, però, le equazioni risultanti non sono lineari, e non possono quindi essere risolte con la procedura LINFIT.

Eliminazione del coefficiente A

Fortunatamente in questo caso il coefficiente A è lineare, e può essere quindi separato; con opportune manipolazioni, la prima delle due equazioni dà:

$$A = \Sigma(e^{Bx} / y) / \Sigma(e^{2Bx} / y^2)$$

Sostituendo questa espressione di A nella seconda equazione si ottiene:

$$\Sigma(e^{Bx} / y)\, \Sigma(xe^{2Bx} / y^2) - \Sigma(e^{2Bx} / y^2)\, \Sigma(xe^{Bx} / y) = 0 = f(B)$$

Avendo eliminato A, questa equazione è funzione solo di B. Nel Capitolo 8 abbiamo sviluppato un programma per la risoluzione di equazioni non lineari con il metodo di Newton; possiamo utilizzarlo qui.

Soluzione con il metodo di Newton

Adesso calcoliamo la derivata dell'equazione precedente rispetto a B; otteniamo

$$\begin{aligned} f'(B) = {} & 2\Sigma(e^{Bx} / y)\, \Sigma(x^2e^{2Bx} / y^2) \\ & - \Sigma(xe^{2Bx} / y^2)\, \Sigma(xe^{Bx} / y) \\ & - \Sigma(e^{2Bx} / y^2)\, \Sigma(x^2e^{Bx} / y) \end{aligned}$$

Prima di proseguire, controlliamo che tutti i termini di f(B) e f'(B) siano omogenei; ogni termine di f(B) deve corrispondere a

$$\frac{x}{y^3}$$

e ogni termine di f'(B) a

$$\frac{x^2}{y^3}$$

PROGRAMMA PASCAL: INTERPOLAZIONE NON LINEARE DELLA CURVA ESPONENZIALE

Il programma della Figura 10.5 si può usare per trovare un'interpolazione non lineare con i minimi quadrati per l'equazione di diffusione:

$$D = D_0 e^{-Q/RT}$$

che abbiamo già visto. Poichè stiamo usando il metodo di Newton, abbiamo bisogno di una prima approssimazione per il valore di B; possiamo ottenerlo dall'equazione linearizzata, ma invece di fare una completa interpolazione lineare, possiamo semplicemente calcolare il valore di B dall'equazione 20 del Capitolo 5. In questo caso, y viene sostituito da ln y.

$$B = \frac{\Sigma[x \ln(y)] - \Sigma(x)\,\Sigma[\ln(y)/n]}{\Sigma x^2 - (\Sigma x)^2/n}$$

Le istruzioni per questa prima approssimazione sono nella procedura NLIN; un approccio più complicato potrebbe consistere nel richiamare la procedura di Gauss Jordan (Figura 4.5) per la prima approssimazione.

Esecuzione del programma

Scrivete il programma ed eseguitelo; confrontate i risultati con la Figura 10.6.

```
PROGRAM NLIN3(OUTPUT);
  (* 23 Gen 81 *)

(* Programma pascal che esegue una *)
(* interpolazione non lineare coi minimi quadrati *)
(* per la diffusione dello Zn nel Cu *)

CONST
  MAXR = 20 (* data points *);
  MAXC = 4 (* termini polinomiali *);
  R    = 1.987 (* costante del gas *);

TYPE
  INDEX = 1..MAXR;
  ARY   = ARRAY[INDEX] OF REAL;
  ARYS  = ARRAY[1..MAXC]  OF REAL;
  ARY2  = ARRAY[1..MAXR, 1..MAXC] OF REAL;

VAR
  X, Y, Y_CALC, T, D, EX : ARY;
  COEF          : ARYS (* vettore soluzione *);
  I, N          : INTEGER;
  NROW, NCOL    : INTEGER;
  DONE, ERROR   : BOOLEAN;
  CORREL_COEF, A, B, X2, SRS : REAL;

PROCEDURE GET_DATA(VAR X, Y : ARY;
                   VAR N: INTEGER);

(* riceve i valori per n e le matrici t, d *)

VAR
  I : INTEGER;
```

Figura 10.5 — Interpolazione con i minimi quadrati per una funzione esponenziale non linearizzata.

```
BEGIN
  N := 7;
  T[1] := 600.0; D[1] := 1.4E-12;
  T[2] := 650.0; D[2] := 5.5E-12;
  T[3] := 700.0; D[3] := 1.8E-11;
  T[4] := 750.0; D[4] := 6.1E-11;
  T[5] := 800.0; D[5] := 1.6E-10;
  T[6] := 850.0; D[6] := 4.4E-10;
  T[7] := 900.0; D[7] := 1.2E-9;
  FOR I := 1 TO N DO
    BEGIN
      X[I] := 1.0/(T[I] + 273);
      Y[I] := D[I]
    END
END (* procedure get_data *);

PROCEDURE WRITE_DATA;

(* stampa i risultati *)

VAR
  I : INTEGER;

BEGIN
  WRITELN;
  WRITELN('  I   T C   D ',
          '  D Calc');
  FOR I := 1 TO N DO
    WRITELN(I:3, T[I]:8:0, D[I],
          ' ', Y_CALC[I]);
  WRITELN;
  WRITELN('Coefficienti');
  WRITELN(COEF[1], '  Termine costante');
```

Figura 10.5 — Interpolazione con i minimi quadrati per una funzione esponenziale non linearizzata (segue).

```
    FOR I := 2 TO NCOL DO
      WRITELN(COEF[I]) (* altri termini *);
    WRITELN;
    WRITELN(' D0 = ', A:7:2,' cm sq/sec.');
    WRITELN(' Q = ', -R * B/1000 :8:2, ' kcal/mole');
    WRITELN; WRITELN(' SRS = ', SRS :8:4)
  END (* write_data *);

  PROCEDURE FUNC(B : REAL;
          VAR FB, DFB : REAL);

  VAR
    I : INTEGER;
    S1, S2, S3, S4, S5, S6,
    EX1, EX2, XI, X2, YI, Y2 : REAL;

  BEGIN
    S1 := 0.0;
    S2 := 0.0;
    S3 := 0.0;
    S4 := 0.0;
    S5 := 0.0;
    S6 := 0.0;
    FOR I:= 1 TO N DO
      BEGIN
        XI := X[I];
        X2 := XI * XI;
        YI := Y[I];
        Y2 := YI * YI;
        EX1 := EXP(B * XI);
        EX[I] := EX1;
        EX2 := EX1 * EX1;
        S1 := S1 + XI * EX2/Y2;
```

Figura 10.5 — Interpolazione con i minimi quadrati per una funzione esponenziale non linearizzata (segue).

```
            S2 := S2 + EX1/YI;
            S3 := S3 + XI * EX1/YI;
            S4 := S4 + EX2/Y2;
            S5 := S5 + 2.0 * X2 * EX2/Y2;
            S6 := S6 + X2 * EX1/YI
         END;
      FB := S1 * S2 - S3 * S4;
      DFB := S2 * S5 - S1 * S3 - S4 * S6;
      A := S2/S4
   END (* func *);

   PROCEDURE NEWTON
               (VAR X: REAL);
   (* 23 Gen 81 *)
   CONST
      TOL = 1.0E-6;
      MAX = 20;
   VAR
      FX, DFX : REAL;
      DX, X1 : REAL;
      I : INTEGER;

   BEGIN (* newton *)
      ERROR := FALSE;
      I := 0;
      REPEAT
         I := I + 1;
         X1 := X;
         FUNC(X, FX, DFX);
         IF DFX = 0.0 THEN
            BEGIN
               ERROR := TRUE;
               X := 1.0;
               WRITELN ('ERRORE: pendenza zero')
            END
```

Figura 10.5 — Interpolazione con i minimi quadrati per una funzione esponenziale non linearizzata (segue).

```
      ELSE
         BEGIN
            DX := FX/DFX;
            X := X1 - DX;
         END
   UNTIL
      ERROR OR
      (I > MAX) OR
      (ABS(DX) <= ABS(TOL * X));
   IF I > MAX THEN
      BEGIN
         WRITELN (´ERRORE: non convergente in´,
               MAX, ´cicli´);
         ERROR := TRUE
      END
END (* newton *);
PROCEDURE NLIN(X, Y : ARY;
         VAR Y_CALC : ARY;
                     N : INTEGER);
(* approssima l'equazione di diffusione attraverso
   n coppie di punti x e y *)
VAR
   RESID : ARY;
   MATR : ARY2;
   I      : INTEGER;
   XI, YI, SUM_X,
   SUM_Y, SUM_Y2, B1,
   SUM_XY, SUM_X2 : REAL;
BEGIN (* nlin *)
   NCOL := 2 (* due termini *);
   SUM_X := 0.0;
   SUM_Y := 0.0;
   SUM_XY := 0.0;
```

Figura 10.5 — Interpolazione con i minimi quadrati per una funzione esponenziale non linearizzata (segue).

```
    SUM_X2 := 0.0;
    FOR I := 1 TO N DO
      BEGIN
        XI := X[I];
        YI := LN(Y[I]);
        SUM_X := SUM_X + XI;
        SUM_Y := SUM_Y + YI;
        SUM_Y2 := SUM_Y2 + YI * YI;
        SUM_XY := SUM_XY + XI * YI;
        SUM_X2 := SUM_X2 + XI * XI
      END;

    B := (SUM_XY - SUM_X * SUM_Y/N) /
         (SUM_X2 - SQR(SUM_X)/N);
    NEWTON(B);
    COEF[1] := A;
    COEF[2] := B;
    SRS := 0.0;
    FOR I := 1 TO N DO
      BEGIN
        Y_CALC[I] := A * EX[I];
        IF Y[I] <> 0.0 THEN
          RESID[I] := Y_CALC[I]/Y[I] - 1:0
        ELSE RESID[I] := Y[I]/Y_CALC[I] - 1.0;
        SRS := SRS + SQR(RESID[I])
      END
  END (* nlin *);

  BEGIN (* programma principale *)
    GET_DATA (X, Y, N);
    NLIN(X, Y, Y_CALC, N);
    WRITE_DATA
  END.
```

Figura 10.5 — Interpolazione con i minimi quadrati per una funzione esponenziale non linearizzata (segue).

```
I     T C        D              D Calc
1     600.  1.40000e-12     1.31051e-12
2     650.  5.50000e-12     5.41377e-12
3     700.  1.80000e-11     1.93304e-11
4     750.  6.10000e-11     6.09471e-11
5     800.  1.60000e-10     1.72657e-10
6     850.  4.40000e-10     4.45808e-10
7     900.  1.20000e-9      1.06167e-9

Coefficienti
3.08933e-1          Termine costanti
-2.28603e4

D0 =     0.31 cm sq/sec.
Q =     45.42 kcal/mole

SRS =  0.0295
```

Figura 10.6 — La diffusione dello zinco nel rame (interpolazione non lineare).

Ora confrontate i risultati della Figura 10.4 con quelli della Figura 10.6: i valori della costante di diffusione D_0 e dell'energia di attivazione Q sono circa gli stessi, mentre la somma dei quadrati dei residui, SRS, è minore nell'interpolazione non lineare. Precisiamo però che, se si fossero usati i residui linearizzati

$$r = \ln D_0 - \frac{Q}{RT}$$

per il calcolo di SRS, il valore per l'interpolazione lineare sarebbe più piccolo. Inoltre, se i dati seguissero esattamente un'equazione esponenziale, avremmo esattamente gli stessi risultati in entrambi i casi; perciò conviene eseguire sia l'interpolazione lineare che quella non lineare. Se i risultati sono molto diversi, è probabile che si sia fatto un errore nella misurazione o nella registrazione dei dati.

SOMMARIO

Abbiamo visto esempi di due tecniche di interpolazione non lineare, con la funzione razionale (1) e la funzione esponenziale (2); abbiamo usato un approccio di linearizzazione e uno diretto. È importante ricordare che nessuno dei due rappresenta un metodo generale di interpolazione non lineare, ma sono tecniche particolari da usare con particolari equazioni.

CAPITOLO 11

APPLICAZIONI AVANZATE

La Curva Normale, la Funzione di errore Gaussiana, la Funzione Gamma, la Funzione di Bessel

INTRODUZIONE

Questo capitolo affronta alcune applicazioni avanzate della programmazione alla matematica, usando strumenti che abbiamo messo a punto nei capitoli precedenti del libro. Studieremo tre funzioni che hanno importanti applicazioni in matematica, in fisica e in ingegneria: la funzione di errore Gaussiana, la funzione Gamma e le funzioni di Bessel.

Scriveremo due programmi per valutare la funzione di errore Gaussiana; il primo usa la regola di Simpson per l'integrazione numerica, il secondo uno sviluppo in serie di infiniti termini. Nel discutere la funzione Gaussiana riprenderemo in considerazione il problema della diffusione, già affrontato nel Capitolo 10 a proposito dell'interpolazione non lineare.

Studieremo poi la funzione Gamma, e svilupperemo un programma per valutarla in base alle sue particolari proprietà. Useremo infine questo programma per l'ultimo argomento del capitolo, una ricerca di soluzioni numeriche per l'equazione di Bessel, del primo e del secondo tipo.

La precisione e il campo di applicabilità del vostro compilatore Pascal sono determinanti nell'esecuzione dei programmi di questo capitolo; perciò sarà necessario richiamare i risultati dei programmi di valutazione eseguiti nel Capitolo 1.

Cominciamo col rivedere i concetti fondamentali delle funzioni di distribuzione.

LE FUNZIONI DI DISTRIBUZIONE NORMALE E CUMULATIVA

Abbiamo visto nel Capitolo 2 che gli errori casuali che capitano durante misurazioni sperimentali producono una distribuzione dei valori misurati intorno al valor

medio. Il grafico della frequenza dei dati risultanti ha la forma di una campana, che prende il nome di distribuzione normale o funzione della densità di probabilità, come si vede nella figura 11.1.

La funzione normale è definita dalla equazione:

$$f(x) = \frac{e^{\frac{-x^2}{2}}}{\sqrt{2\pi}} \tag{1}$$

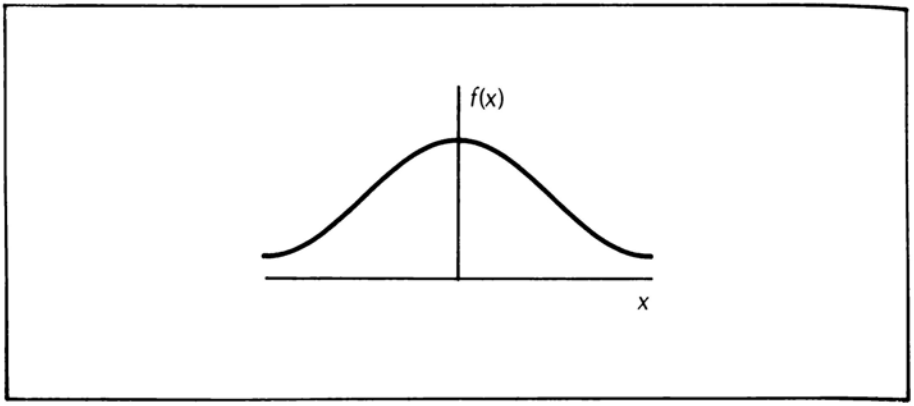


Figura 11.1 — La funzione normale di distribuzione della probabilità.

Questa funzione presenta un picco, corrispondente al valor medio, per x=0, e varia da meno infinito a più infinito. L'area sottesa dalla curva (sopra l'asse delle x) è normalizzata a 1, cioè:

$$\int_{-\infty}^{\infty} f(x)dx = 1 \tag{2}$$

Data la simmetria della curva, è facile vedere che l'area da 0 a infinito, cioè sottesa dalla metà destra della curva, è uguale a metà dell'area totale.

L'area della figura compresa tra $x = -\infty$ e $x = b$ prende il nome di funzione di distribuzione cumulativa F(x).

$$F(x) = \int_{-\infty}^{b} f(x)dx \tag{3}$$

Questo integrale non può essere risolto in questa forma, ma il suo valore è riportato nelle tavole dei manuali. Tavolta viene riportato al suo posto l'integrale

$$G(x) = \int_{0}^{b} f(x)dx \tag{4}$$

Siccome la curva è normalizzata, un integrale può essere ottenuto facilmente dall'altro, in base alla relazione:

$$F(x) = G(x) + 0.5$$

Nel prossimo paragrafo studieremo alcuni metodi di calcolo numerico per ottenere G(x), ma prima approfondiremo lo studio della curva normale.

La deviazione standard

L'area sottesa dalla curva normale è funzione della deviazione standard. Una piccola deviazione standard corrisponde a un insieme di valori misurati raggruppati vicino al valor medio, mentre una grande deviazione standard corrisponde a un insieme di valori distribuiti lontano dalla media. La Figura 11.2 mostra due distribuzioni normali, con lo stesso valor medio ma con diversa deviazione standard. La curva con deviazione standard minore ha un picco più alto per x = 0, ma l'area sottesa dalle due curve è uguale.

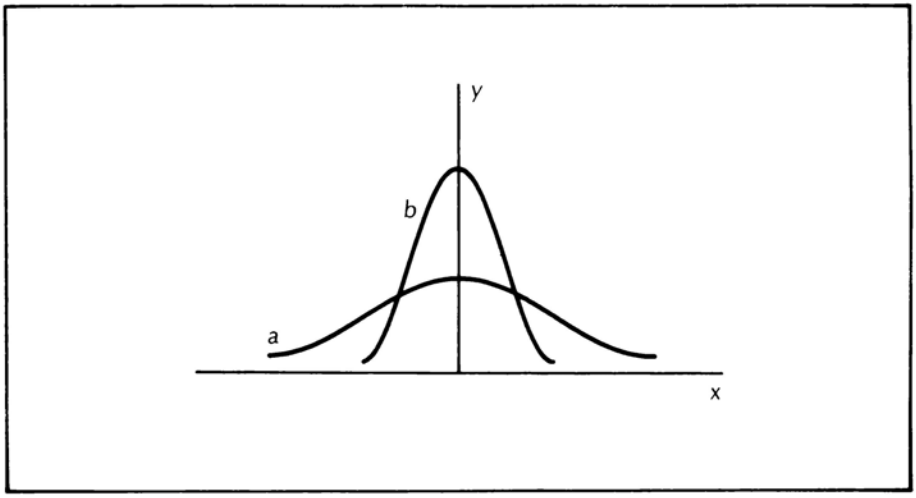


Figura 11.2 — Curve normali con deviazione standard grande (a) e piccola (b).

La Figura 11.3 mostra un grafico della funzione di distribuzione descritta dall'equazione 3. Questa funzione assume il valore 0.5 per x = 0, e tende asintoticamente a 1 per x tendente a più infinito.

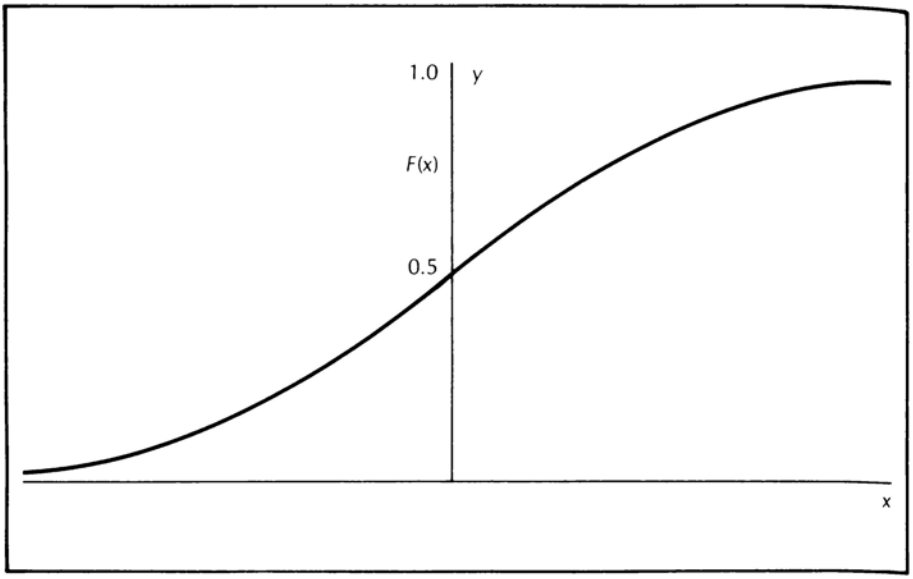


Figura 11.3 — La funzione di distribuzione cumulativa.

La funzione G(x) può essere usata per trovare la relazione tra la deviazione standard e la corrispondente frazione di un particolare campione. Per esempio, G(1) vale 0.34, cioè il 34% della popolazione è nell'intervallo da 0 a 1 della deviazione standard. Il doppio di questo valore, il 68%, corrisponde a un intervallo di deviazione standard uguale a 1 da entrambe le parti del valor medio. La funzione di distribuzione si ottiene facilmente dalla funzione Gaussiana, che discuteremo nel prossimo paragrafo.

LA FUNZIONE DI ERRORE GAUSSIANA

Prima di definire la formula della funzione di errore Gaussiana e di sviluppare un programma per valutarla, faremo una breve digressione su un campo di applicazione che ci è famigliare — la diffusione. L'equazione che descrive la diffusione di un tipo di atomi in un altro tipo ci porterà infine alla nostra funzione di errore Gaussiana.

Diffusione in una sbarra monodimensionale

Per diffusione si intende il flusso di atomi, elettroni o calore da una regione di alta

concentrazione ad una di concentrazione minore. Il flusso risultante, J, può essere descritto dalla prima legge di Fick:

$$J = -D \frac{dC}{dx}$$

dove C è la concentrazione e x la distanza. Il coefficiente di diffusione, o diffusività, D, è la stessa quantità che abbiamo usato per descrivere la diffusione dello zinco nel rame, nel Capitolo 10. Il segno meno significa che il flusso si ha nella direzione opposta al gradiente della concentrazione. Se si tratta di un flusso di atomi, J si esprime in unità di atomi per unità di superficie al secondo. La concentrazione, C, si esprime in unità di atomi per unità di volume.

La prima legge di Fick si può usare anche per descrivere il flusso degli elettroni in un conduttore. Possiamo scrivere:

$$J = \sigma \frac{dV}{dx}$$

dove J è la densità della corrente, V la tensione e x la distanza; σ è la conduttività elettrica.

Analogamente, il flusso di calore si può esprimere come:

$$J = k \frac{dT}{dx}$$

dove J è la conduttività termica in unità di energia per unità di superficie al secondo, e T è la temperatura.

La seconda legge di Fick

$$\frac{\partial C}{\partial t} = D \frac{\partial^2 C}{\partial x^2}$$

può essere usata per descrivere la concentrazione in funzione della posizione e del tempo. Esistono molte diverse soluzioni della seconda legge di Fick, che dipendono dalle condizioni al contorno. Per esempio, quando la concentrazione non cambia più al passare del tempo, si ha una condizione stazionaria. Poichè

$$\frac{\partial C}{\partial t} = 0$$

si ha

$$\frac{D \partial^2 C}{\partial x^2} = 0$$

Questo significa che il gradiente della concentrazione ha un andamento rettilineo.

Un'altra utile soluzione descrive la diffusione di un tipo di atomi in un altro. La concentrazione superficiale degli atomi che si diffondono è costante al variare del tempo; supponiamo che gli altri atomi si presentino come una sbarra monodimensionale, semi-infinita. La soluzione, in questo caso è:

$$\frac{C_x - C_0}{C_s - C_0} = 1 - \mathrm{erf}(y) = 1 - \mathrm{erf}\left(\frac{x}{2\sqrt{Dt}}\right)$$

dove C_x è la concentrazione degli atomi in diffusione al tempo t e alla distanza x dalla superficie, C_s è la concentrazione superficiale, costante al variare del tempo, e C_0 è la concentrazione iniziale uniforme per ogni x, al tempo t = 0. D è il coefficiente di diffusione, e y ha il valore

$$\frac{x}{2\sqrt{Dt}}$$

Se la concentrazione iniziale C_0 è zero, l'equazione diventa

$$\frac{C_x}{C_s} = 1 - \mathrm{erf}(y) = 1 - \mathrm{erf}\left(\frac{x}{2\sqrt{Dt}}\right)$$

dove erf è la funzione di errore Gaussiana, definita come

$$\mathrm{erf}(y) = \frac{2}{\sqrt{\pi}} \int_0^y e^{-t^2} dt \qquad (5)$$

La funzione F(x), descritta dall'equazione 3, e la G(x), descritta dalla 4, possono essere ricavate dalla funzione di errore tramite le relazioni:

$$F(x) = \frac{1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

$$G(x) = \frac{\mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)}{2}$$

Per esempio, l'intervallo rappresentato da due deviazioni standard da entrambe le parti rispetto al valor medio, si può ricavare dalla funzione di errore:

$$2G(2) = \mathrm{erf}\left(\frac{2}{\sqrt{2}}\right) = 95\%$$

PROGRAMMA PASCAL: CALCOLO DELLA FUNZIONE DI ERRORE GAUSSIANA CON LA REGOLA DI SIMPSON

La funzione di errore non può essere calcolata in forma chiusa, ma è possibile ottenerne soluzioni particolari. Un approccio immediato è quello di usare una tecnica di integrazione numerica, come la regola di Simpson. In Figura 11.4 è mostrato un programma per questo.

```
PROGRAM ERFSIMP(INPUT, OUTPUT);

(* integrazione con il metodo di Simpson *)
(* 9 Feb 81 *)

CONST
   TOL = 1.0E-5;
   PI  = 3.141593;

VAR
   DONE : BOOLEAN;
   SUM, UPPER, LOWER, ERF,
   TWOPI : REAL;

FUNCTION FX(X : REAL) : REAL;

BEGIN
   FX := EXP(-X * X)
END; (* function fx *)

PROCEDURE SIMPS(LOWER, UPPER, TOL : REAL;
                                VAR SUM : REAL);

(* integrazione numerica con la regola di Simpson *)
(* la funzione è fx, gli estremi sono lower ed upper *)
(* con un numero di regioni uguale a pieces *)
(* la partizione è delta_x, il risultato è SUM *)
```

Figura 11.4 — La funzione di errore Gaussiana con la regola di Simpson.

```
VAR
   I : INTEGER;
   X, DELTA_X, EVEN_SUM,
   ODD_SUM, END_SUM, SUM1 : REAL;
   PIECES : INTEGER;

BEGIN
   PIECES := 2;
   DELTA_X := (UPPER - LOWER)/PIECES;
   ODD_SUM := FX(LOWER + DELTA_X);
   EVEN_SUM := 0.0;
   END_SUM := FX(LOWER) + FX(UPPER);
   SUM := (END_SUM + 4.0 * ODD_SUM) * DELTA_X/3.0;
   REPEAT
      PIECES := PIECES * 2;
      SUM1 := SUM;
      DELTA_X := (UPPER - LOWER)/PIECES;
      EVEN_SUM := EVEN_SUM + ODD_SUM;
      ODD_SUM := 0.0;
      FOR I:= 1 TO PIECES DIV 2 DO

         BEGIN
            X := LOWER + DELTA_X * (2.0 * I - 1.0);
            ODD_SUM := ODD_SUM + FX(X)
         END;
      SUM := (END_SUM + 4.0 * ODD_SUM
               + 2.0 * EVEN_SUM) * DELTA_X/3.0
   UNTIL ABS(SUM1 - SUM) <= ABS(TOL * SUM1)
END (* simps *);
```

Figura 11.4 — La funzione di errore gaussiana con la regola di Simpson (segue).

```
BEGIN (* programma principale *)
   DONE := FALSE;
   TWOPI := 2.0/SQRT(PI);
   LOWER := 0.0;

   REPEAT
      WRITELN;
      WRITE('Erf di 0.0 è 0.0')
      READLN(UPPER);
      IF UPPER < 0.0 THEN DONE := TRUE
      ELSE IF UPPER = 0.0 THEN
         WRITELN('Erf di 0.: è 0.0')

      ELSE (* upper > 0 *)
         BEGIN
            SIMPS(LOWER, UPPER, TOL, SUM);
            ERF := TWOPI * SUM;
            WRITELN('Erf di', UPPER:7:2,
               ',è', ERF:8:4)
         END
   UNTIL DONE
END.
```

Figura 11.4 — La funzione di errore gaussiana con la regola di Simpson (segue).

Esecuzione del programma

Il programma ripete più volte lo stesso ciclo, chiedendo all'utente dei dati di ingresso; la funzione di errore è calcolata con il metodo di Simpson, e vengono stampati l'argomento e il corrispondente valore della funzione. Il programma ha termine quando si introduce un valore negativo. Ci sono diversi svantaggi nel calcolare la funzione di errore in questo modo; all'aumentare del valore dell'argomento, aumenta il tempo di esecuzione e diminuisce la precisione del calcolo. Per un compilatore con aritmetica in floating point fino a sei o sette cifre di precisione, l'argomento può variare da zero a tre.

PROGRAMMA PASCAL: CALCOLO DELLA FUNZIONE DI ERRORE GAUSSIANA CON UNO SVILUPPO IN SERIE DI INFINITI TERMINI

Un altro modo per calcolare la funzione di errore è di svilupparla in una serie di infiniti termini, e di integrarla poi termine a termine. Il risultato è

$$\mathrm{erf}(y) = \frac{2}{\sqrt{\pi}} \; e^{-y^2} \sum_{n=0}^{\infty} \frac{2^n y^{2n+1}}{1 \cdot 3 \cdot \ldots \cdot (2n+1)}$$

Nella Figura 11.5 c'è un programma per calcolare la funzione di errore in questo modo. Scrivetelo, compilatelo ed eseguitelo; chiederà all'utente di introdurre un valore per l'argomento, e stamperà quindi questo valore e il valore corrispondente della funzione.

La serie infinita è calcolata nella procedura ERF, aggiungendo alla somma ogni nuovo termine. Quando un certo termine non modifica la somma per più del valore TOL, la routine ha termine.

```
PROGRAM ERFD(INPUT, OUTPUT);

(* valutazione della funzione d'errore Gaussiana *)
(* 23 Gen 81 *)

VAR
   X, ANS : REAL;
   DONE : BOOLEAN;

FUNCTION ERF(X : REAL) : REAL;

(* Sviluppo della serie infinita della
   funzione d'errore Gaussiana *)

CONST
   SQRTPI = 1.7724538;
   TOL = 1.0E-6;

VAR
   X2, SUM, SUM1, TERM : REAL;
   I : INTEGER;
```

Figura 11.5 — Uno sviluppo in serie infinita per la funzione di errore Gaussiana.

```
BEGIN
  IF X = 0.0 THEN ERF := 0.0
  ELSE IF X > 4.0 THEN ERF := 1.0
  ELSE
    BEGIN
      X2 := X * X;
      SUM := X;
      TERM := X;
      I := 0;
      REPEAT
        I := I + 1;
        SUM1 := SUM;
        TERM := 2.0 * TERM * X2/(1.0 + 2.0 * I);
        SUM := TERM + SUM1
      UNTIL TERM < TOL * SUM;
      ERF := 2.0 * SUM * EXP(-X2)/SQRTPI
    END (* IF *)
END (* erf *);

BEGIN (* programma principale *)
  DONE := FALSE;
  WRITELN;
  REPEAT
    WRITE(' Arg? ');
    READLN(X);
    IF X < 0.0 THEN DONE := TRUE
    ELSE
      BEGIN
        ANS := ERF(X);
        WRITELN ('Erf di', X :6:3,
          'è', AND :9:5;
      END
  UNTIL DONE
END.
```

Figura 11.5 — Uno sviluppo in serie infinita per la funzione di errore Gaussiana (segue).

Esecuzione del programma

L'utente deve introdurre per l'argomento un valore maggiore o uguale a zero; la funzione risultante varia da zero a uno, e precisamente è zero se l'argomento è zero, e si avvicina all'unità per valori dell'argomento maggiori di 4. D'altra parte, nell'intervallo da 0 a 0.6, la funzione è circa uguale all'argomento. La funzione di errore ha la stessa forma della funzione di distribuzione cumulativa di Figura 11.3.

In Figura 11.6 vi diamo alcuni valori della funzione stessa. Il programma continua a eseguire un ciclo, dando nuovi valori della funzione di errore fin quando viene introdotto un valore negativo per l'argomento.

y	*erf(y)*
0.0	0.0
0.1	0.1125
0.2	0.2227
0.3	0.3286
0.4	0.4284
0.5	0.5205
0.7	0.6778
1.0	0.8427
2.0	0.9953

Figura 11.6 — La funzione di errore Gaussiana.

Supponiamo di studiare la diffusione dello zinco in una sbarra di rame a 900° C. Possiamo usare la funzione di errore per determinare la concentrazione del rame in funzione del tempo e della distanza dalla superficie. Il coefficiente di diffusione può essere calcolato dalla costante di diffusione e dall'energia di attivazione, che sono state calcolate nel caso dell'interpolazione non lineare nel Capitolo 10.

$$D = 0.31\, e^{\frac{-45{,}420}{1.987(900+273)}}$$

Per una distanza di 0.01 cm e un tempo di 13 ore, l'argomento y della funzione di errore vale 0.7, la funzione di errore vale 0.67, e il complemento della funzione di errore 0.33. Questo significa che la concentrazione dello zinco a una profondità di 0.01 cm sarà il 33% della concentrazione superficiale dopo 13 ore.

Nel prossimo paragrafo vedremo che gli errori di arrotondamento rendono difficile il calcolo di una soluzione diretta del complemento della funzione di errore, e studieremo un programma che usa due funzioni, per trattare sia argomenti grandi che piccoli.

IL COMPLEMENTO DELLA FUNZIONE DI ERRORE

Nella precedente soluzione dell'equazione di diffusione, ci interessava in realtà il complemento della funzione di errore più che la funzione stessa. Il complemento è definito come:

$$\mathrm{erfc}(y) = \frac{2}{\sqrt{\pi}} \int_{y}^{\infty} e^{-t^2} dt \tag{6}$$

e si ottiene dalla relazione:

$$\mathrm{erfc}(y) = 1 - \mathrm{erf}(y)$$

Se però lo si calcola in questo modo, si hanno errori di arrotondamento significativi per valori dell'argomento maggiori di 3. Man mano che erf si avvicina a 1, (1 − erf) si avvicina a zero, e alla fine si perdono tutte le cifre significative. Inoltre il tempo di esecuzione aumenta all'aumentare del valore dell'argomento.

D'altra parte l'equazione 6 non può essere integrata con la regola trapezoidale o con il metodo di Simpson, in quanto è problematico stabilire l'estremo superiore dell'integrale: un valore superiore a 8 darà un underflow, in quanto e^{-64} è troppo piccolo, ma l'area da questo punto all'infinito è significativa e non può essere ignorata.

PROGRAMMA PASCAL: CALCOLO DEL COMPLEMENTO DELLA FUNZIONE DI ERRORE

Una soluzione di questo problema può essere quella di usare due funzioni, una per valori piccoli e una per valori grandi dell'argomento. Il programma della Figura 11.7 usa questo metodo. Lo sviluppo in serie di erf, in Figura 11.5, si usa per i valori più piccoli, mentre per quelli più grandi la funzione si calcola con uno sviluppo asintotico. Questo algoritmo è tanto più preciso quanto più grande è il valore dell'argomento. L'equazione, espressa sotto forma di frazione, invece che con il solito sviluppo in serie, è:

$$\mathrm{erfc}(y) = \frac{1/[1 + v/\{1 + 2v/[1 + 3v/(1 + \ldots)]\}]}{\sqrt{\pi}\, y e^{y^2}}$$

dove

$$v = \frac{1}{2y^2}$$

```
PROGRAM ERFD3(INPUT, OUTPUT);

(* valutazione della funzione d'errore Gaussiana *)
(* 30 Gen 81 *)

VAR
   X, ER, EC : REAL;
   DONE : BOOLEAN;

FUNCTION ERF(X : REAL) : REAL;

(* Sviluppo della serie infinita della
   funzione d'errore Gaussiana *)

CONST
   SQRTPI = 1.7724538;
   TOL = 1.0E-5;

VAR
   X2, SUM, SUM1, TERM : REAL;
   I : INTEGER;

BEGIN
   X2 := X * X;
   SUM := X;
   TERM := X;
   I := 0;
   REPEAT
      I := I + 1;
      SUM1 := SUM;
      TERM := 2.0 * TERM * X2/(1.0 + 2.0 * I);
      SUM := TERM + SUM1
   UNTIL TERM < TOL * SUM;
   ERF := 2.0 * SUM * EXP(-X2)/SQRTPI
END (* erf *);
```

Figura 11.7 — La funzione di errore e il suo complemento.

```
FUNCTION ERFC(X : REAL) : REAL;
(* Serie completa della funzione d'errore *)
CONST
   SQRTPI = 1.7724538;
   TERMS = 12;
VAR
   X2, U, V, SUM : REAL;
   I : INTEGER;
BEGIN
   X2 := X * X;
   V := 1.0/(2.0 * X2);
   U := 1.0 + V * (TERMS + 1.0);
   FOR I := TERMS DOWNTO 1 DO
      BEGIN
         SUM := 1.0 + I * V/U;
         U := SUM
      END;
    ERFC := EXP(-X2)/(X * SUM * SQRTPI)
  END (* ercf *);

BEGIN (* programma principale *)
   DONE := FALSE;
   WRITELN;
   REPEAT
      WRITE(' Arg? ');
      READLN(X);
      IF X < 0.0 THEN DONE := TRUE
      ELSE
         BEGIN
            IF X = 0.0 THEN
```

Figura 11.7 — La funzione di errore e il suo complemento (segue).

```
            BEGIN
              ER := 0.0;
              EC := 1.0
            END
          ELSE
            BEGIN
              IF X < 1.5 THEN
                BEGIN
                  ER := ERF(X);
                  EC := 1.0 - ER
                END
              ELSE
                BEGIN
                  EC := ERFC(X);
                  ER := 1.0 - EC
                END (* IF *)
            END;
          WRITELN(' X= ', X:6:2, ', Erf= ',
            ER :7:4, ', Erfc= ', EC)
        END (* IF *)
    UNTIL DONE
  END.
```

Figura 11.7 — La funzione di errore e il suo complemento (segue).

Esecuzione del programma

Scrivete la nuova versione ed eseguitela. Il programma stamperà la funzione di errore e il suo complemento. Per valori dell'argomento minori di 1.5, la funzione di errore viene calcolata nella procedura ERF con lo sviluppo in serie di infiniti termini di Figura 11.5, e il suo complemento si ottiene con una sottrazione da 1. Se il valore dell'argomento è maggiore o uguale a 1.5, il complemento viene calcolato con lo sviluppo asintotico nella procedura ERFC, e la funzione di errore si ottiene con una sottrazione da 1.

Confrontate la funzione di errore ottenuta con questa versione con i dati della Figura 11.6, e controllate il suo complemento con i dati della Figura 11.8.

y	erfc(y)
1.5	3.390E−2
2.0	4.678E−3
2.5	4.070E−4
3.0	2.209E−5
3.5	7.431E−7
4.0	1.542E−8
4.5	1.966E−10

Figura 11.8 — La funzione di errore complementare.

Ora modificheremo il programma per calcolare la funzione di errore con le parentesi nidificate invece che con un ciclo FOR... DO.

PROGRAMMA PASCAL: UN'IMPLEMENTAZIONE PIÚ VELOCE DEL CALCOLO DELLA FUNZIONE DI ERRORE

Nel programma precedente la funzione di errore veniva calcolata sommando con un ciclo diversi termini; questo metodo iterativo è più lento di una somma diretta, ma ha il vantaggio di funzionare in genere con qualsiasi compilatore Pascal. Ciononostante, quando si può, è preferibile un metodo non iterativo, in cui i termini sono sommati direttamente, come quello che svilupperemo nel prossimo paragrafo.

La funzione di errore viene calcolata nella nuova versione della funzione ERF, come equazione polinomiale del 12° ordine. I coefficienti vengono definiti all'inizio della funzione come T2, T3, T4, etc.. Si è scelto un polinomio a dodici termini poichè tanti sono necessari per un argomento di 1.5; un numero minore basta per argomenti più piccoli, ma, anche così, questo metodo è notevolmente più veloce del precedente.

La funzione di errore complementare è anch'essa calcolata con una somma diretta, e anche in questo caso, il risultato si ottiene più rapidamente.

Con questa nuova versione si può presentare un problema: molti compilatori Pascal non sono in grado di elaborare espressioni con diversi livelli di parentesi. In previsione di ciò, sia l'espressione della funzione di errore che la sua complementare sono state suddivise in due o tre parti, ma anche con questo accorgimento l'elaborazione può risultare troppo complessa per almeno un compilatore. Dovreste comunque tentare almeno di compilare questa versione. Fate una copia del programma precedente e modificate le funzioni secondo la Figura 11.9.

```
PROGRAM ERF4(INPUT, OUTPUT);
(* valutazione della funzione d'errore Gaussiana *)
(* 30 Gen 81 *)
VAR
   X, ER, EC : REAL;
   DONE : BOOLEAN;
FUNCTION ERF(X : REAL) : REAL;
(* Sviluppo della serie infinita della
   funzione d'errore Gaussiana *)
CONST
   SQRTPI = 1.7724538;
   T2 = 0.6666667;
   T3 = 0.2666667;
   T4 = 0.07619048;
   T5 = 0.01693122;
   T6 = 3.078403E-3;
   T7 = 4.736005E-4;
   T8 = 6.314673E-5;
   T9 = 7.429027E-6;
   T10 = 7.820028E-7;
   T11 = 7.447646E-8;
   T12 = 6.476214E-9;
VAR
   X2, SUM : REAL;
BEGIN (* function erf *)
   X2 := X * X;
   SUM := T5 + X2 * (T6 + X2 * (T7 + X2 * (T8 + X2 * (T9
            + X2 * (T10 + X2 * (T11 + X2 * T12))))));
   ERF := 2.0 * EXP(-X2)/SQRTPI
            * (X * (1 + X2 * (T2 + X2 * (T3 + X2 * (T4 + X2
            * SUM)))))
END (* function erf *);
```

Figura 11.9 — Un calcolo non iterativo della funzione di errore.

```
FUNCTION ERFC(X : REAL) : REAL;
(* serie completa della funzione d'errore *)

CONST
   SQRTPI = 1.7724538;

VAR
   X2, V, SUM : REAL;

BEGIN (* function erfc *)
   X2 := X * X;
   V := 1.0/(2.0 * X2);
   SUM := V/(1 + 8 * V/(1 + 9 * V/(1 + 10 * V
            / (1 + 11 * V/(1 + 12 * V)))));
   SUM := V/(1 + 3 * V/(1 + 4 * V/(1 + 5 * V
            / (1 + 6 * V/(1 + 7 * SUM)))));
   ERFC := 1.0/(EXP(X2) * X * SQRTPI
            * (1 + V/(1 + 2 * SUM)))
END (* function ercf *);

BEGIN (* programma principale *)
   DONE := FALSE;
   WRITELN;
   REPEAT
      WRITE(' Arg? ');
      READLN(X);
      IF X < 0.0 THEN DONE := TRUE
      ELSE
         BEGIN
            IF X = 0.0 THEN
               BEGIN
                  ER := 0.0;
                  EC := 1.0
               END
```

Figura 11.9 — Un calcolo non iterativo della funzione di errore (segue).

```
            ELSE
              BEGIN
                IF X < 1.5 THEN
                  BEGIN
                    ER := ERF(X);
                    EC := 1.0 - ER
                  END
                ELSE
                  BEGIN
                    EC := ERFC(X);
                    ER := 1.0 - EC
                  END (* IF *)
              END;
            WRITELN(' X= ', X:6:2, ', Erf= ',
              ER :7:4, ', Erfc= ', EC)
          END (* IF *)
      UNTIL DONE
    END.
```

Figura 11.9 — Un calcolo non iterativo della funzione di errore (segue).

Nel prossimo paragrafo studieremo le caratteristiche particolari della funzione Gamma, e svilupperemo un programma per calcolarla.

LA FUNZIONE GAMMA

Una funzione imparentata con la funzione di errore è nota come la funzione Gamma, ed è definita dall'integrale:

$$\Gamma(n) = \int_0^\infty x^{n-1} e^{-x} dx \tag{7}$$

Questa funzione è importante poichè è una parte della soluzione dell'equazione di Bessel. Inoltre può essere usata per calcolare i fattoriali, con la relazione ricorsiva

$$\Gamma(n+1) = n\Gamma(n) \tag{8}$$

Poichè $\Gamma(1) = 1$, risulta che

$$\begin{aligned}
\Gamma(2) &= \Gamma(1) = 1 &&= 1! \\
\Gamma(3) &= 2\Gamma(2) = 1\cdot 2 &&= 2! \\
\Gamma(4) &= 3\Gamma(3) = 1\cdot 2\cdot 3 &&= 3! \\
&\ldots \\
\Gamma(n) &= (n-1)\Gamma(n-1) &&= (n-1)!
\end{aligned}$$

La formula generale per usare la funzione Gamma per il calcolo dei fattoriali è:

$$\Gamma(n+1) = n\Gamma(n) = \mathrm{n}!$$

Poichè la funzione Gamma è definita per ogni argomento reale maggiore di zero, può essere definito analogamente il fattoriale di argomenti non interi. Inoltre vediamo che

$$\Gamma(0.5) = -0.5! = \sqrt{\pi}$$

e questo sarà utile nel calcolo delle funzioni di Bessel. La funzione Gamma è definita anche per numeri negativi non interi, mentre diventa infinita per un valore dell'argomento nullo o intero negativo. Nella Figura 11.10 è riportato un grafico della funzione Gamma per argomenti reali.

La funzione Gamma può essere calcolata con una forma di approssimazione di Stirling:

$$\Gamma(x) = \sqrt{\frac{2\pi}{x}}\, x^x e^y$$

dove

$$y = \frac{1}{12x} - \frac{1}{360x^3} - x$$

Poichè questa è una serie asintotica, la precisione relativa aumenta all'aumentare del valore dell'argomento.

PROGRAMMA PASCAL: CALCOLO DELLA FUNZIONE GAMMA

Il programma della Figura 11.2 usa l'approssimazione di Stirling per calcolare la funzione Gamma. L'argomento può essere qualsiasi numero reale maggiore di zero e minore di 32. Il limite superiore dipende dall'aritmetica in floating point del vostro compilatore. L'argomento può essere anche negativo, se non è intero, mentre non può essere nullo o intero negativo.

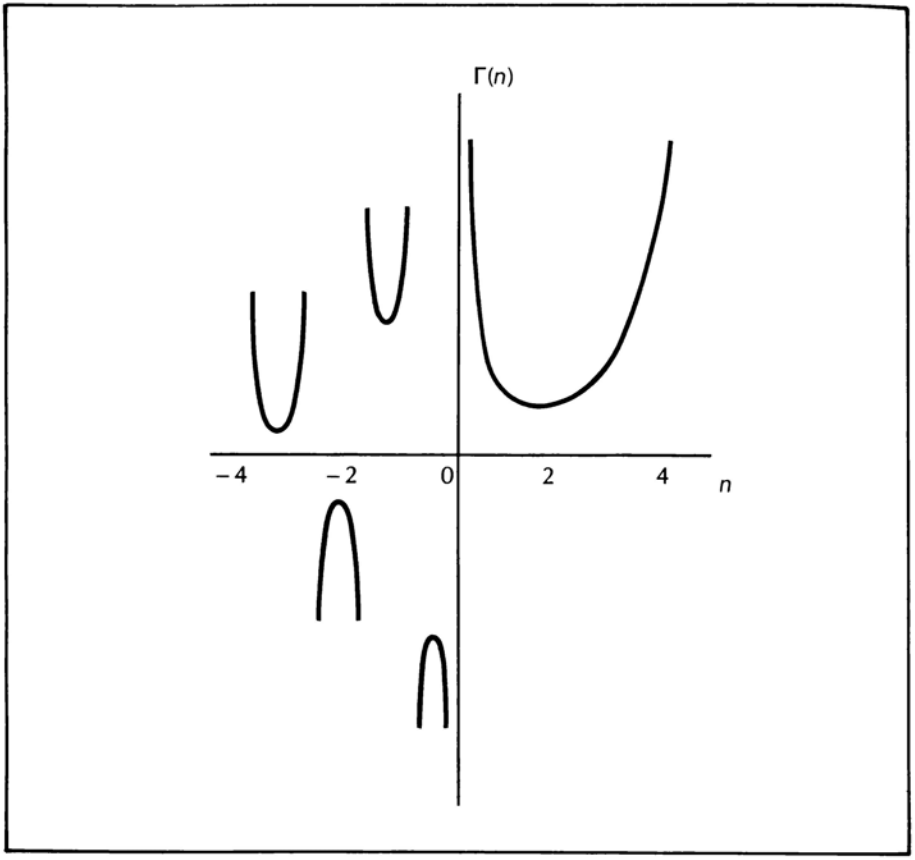


Figura 11.10 — La funzione Gamma.

Gli argomenti positivi vengono incrementati ogni volta di 2, e viene calcolata la funzione Gamma corrispondente al nuovo argomento. Il valore risultante è poi ricondotto all'argomento originale con l'algoritmo:

$$\Gamma(x) = \frac{\Gamma(x+2)}{x(x+1)}$$

Questa conversione non è necessaria per gli argomenti di valore maggiore, ma garantisce sei cifre di precisione per tutti i valori di x.

Gli argomenti negativi sono incrementati fino a diventare positivi, e la funzione Gamma viene richiamata ricorsivamente per calcolare il nuovo valore. Il risultato viene poi corretto per l'argomento originale.

Esecuzione del programma

Scrivete il programma ed eseguitelo. Provate con i dati della Figura 11.11 e confrontate i valori di $\Gamma(x)$ con i vostri risultati. Il programma esegue ripetutamente un ciclo, che ha termine se inserite un numero minore di −22. Poichè:

$$x! = \Gamma(x+1)$$

il programma può essere facilmente riscritto per generare fattoriali invece della funzione Gamma.

x	Γ(x)	
1	1	0!
2	1	1!
3	2	2!
4	6	3!
5	24	4!
6	120	5!
0.5	1.7725	$\sqrt{\pi}$
−0.5	−3.5449	−Γ(0.5) / 0.5
−1.5	2.3633	−Γ(−0.5) / 1.5

Figura 11.11 — Alcuni valori della funzione Gamma.

```
PROGRAM TSTGAM(INPUT, OUTPUT);
 (* prova della funzione gamma *)
 (* 26 Gen 81 *)

VAR
   X : REAL;

FUNCTION GAMMA(X : REAL) : REAL;
 (* 26 Gen 81 *)

CONST
   PI = 3.1415926;

VAR
   I, J : INTEGER;
   Y, GAM : REAL;

BEGIN (* gamma function *)
   IF X >= 0.0 THEN
      BEGIN
         Y := X + 2.0;
         GAM := SQRT(2 * PI/Y)
            * EXP(Y * LN(Y) + (1 - 1/(30 * Y * Y))
            /(12 * Y) - Y);
         GAMMA := GAM/(X * (X + 1))
      END
   ELSE (* x < 0 *)
      BEGIN
         J := 0;
         Y := X;
         REPEAT (* incrementa l'argomento fino a quando è positivo *)
            J := J + 1;
            Y := Y + 1.0
```

Figura 11.12 — Calcolo della funzione Gamma.

```
        UNTIL Y > 0.0;
        GAM := GAMMA(Y) (* chiamata ricorsiva *);
        FOR I := 0 TO J - 1 DO
          GAM := GAM/(X + I);
        GAMMA := GAM
      END (* x< 0 *)
END (* gamma function *);

BEGIN (* programma principale *)
  WRITELN;
  REPEAT
    REPEAT
      WRITE(' X: ');
      READLN(X)
    UNTIL X <> 0.0;
    WRITELN( 'Gamma è', GAMMA(X))
  UNTIL X < -22.0
END.
```

Figura 11.12 — Calcolo della funzione Gamma (segue).

Negli ultimi due paragrafi di questo capitolo introdurremo le funzioni di Bessel del primo e del secondo tipo. L'equazione di Bessel è un'equazione differenziale con molte applicazioni matematiche e scientifiche, e le sue soluzioni si possono trovare usando le funzioni di Bessel. Svilupperemo due programmi Pascal per queste funzioni.

FUNZIONI DI BESSEL

L'equazione di Bessel:

$$x^2y'' + xy' + (x^2 - n^2)y = 0$$

compare nell'analisi di molti problemi di diverso genere che presentano una simme-

tria circolare. In questa equazione, x è la variabile indipendente, y la variabile dipendente e n una costante che prende il nome di ordine. È un'equazione differenziale non lineare, che non può essere risolta in forma chiusa. Una delle sue soluzioni è:

$$y = J_n(x)$$

dove J_n è la funzione di Bessel di ordine n-esimo del primo tipo.

I valori delle funzioni di Bessel sono stati ampliamente tabulati per particolari valori di x e di n, ma sono difficili da usare in questa forma. Per valori di x minori di 15, possono essere calcolate dalla serie di infiniti termini:

$$J_n(x) = \sum_{k=0}^{n} \frac{(-1)^k}{k!\Gamma(n+k+1)} \left(\frac{x}{2}\right)^{n+2k} \tag{9}$$

D'altra parte l'espressione asintotica:

$$J_n(x) = \sqrt{\frac{2}{\pi x}} \cos\left(x - \frac{\pi}{4} - \frac{n\pi}{2}\right) \tag{10}$$

può essere usata per valori di x maggiori. In Figura 11.13 è riportato il grafico delle funzioni di Bessel J_0 e J_1.

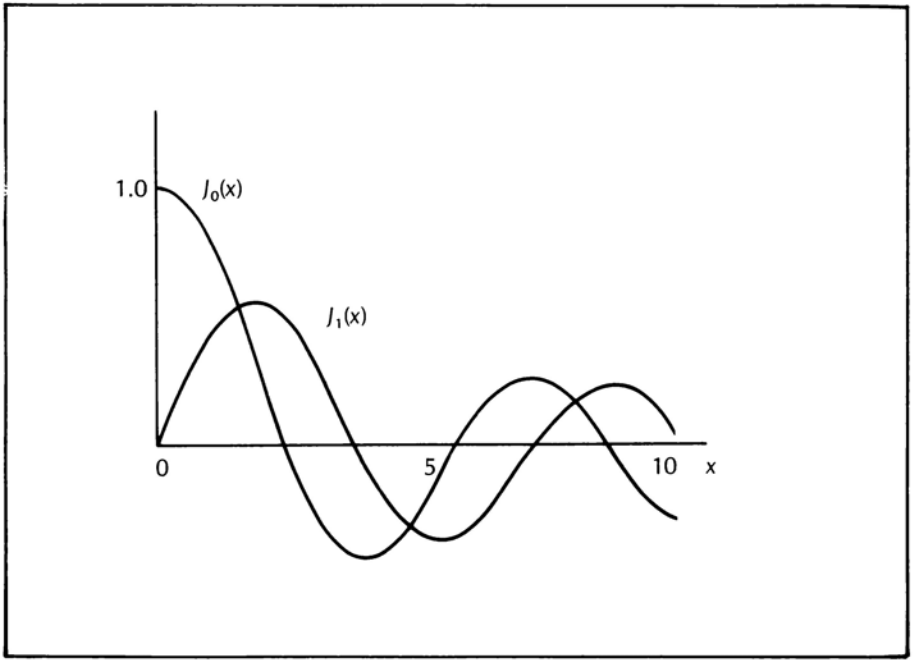


Figura 11.13 — Le funzioni di Bessel J_0 e J_1.

PROGRAMMA PASCAL: FUNZIONI DI BESSEL DEL PRIMO TIPO

Il programma della Figura 11.15 usa entrambe le equazioni 9 e 10 per calcolare le funzioni di Bessel del primo tipo. L'ordine può essere zero o un numero positivo; l'argomento può essere anche un numero negativo non intero. Il programma comprende la funzione Gamma del precedente paragrafo. Per argomenti minori di 15 si usano le serie di infiniti termini, per argomenti maggiori l'espressione asintotica. Ci può essere una caduta di precisione nella regione di passaggio, dovuta alle caratteristiche dell'aritmetica in floating point del vostro compilatore.

Scrivete il programma ed eseguitelo. Provate con i dati di Figura 11.14 per l'ordine e l'argomento, e confrontate i risultati. Il programma ha termine quando inserite un valore minore di −25 per l'ordine.

Poichè l'equazione di Bessel è del secondo ordine, esistono due soluzioni indipendenti. Quando l'ordine n non è intero, entrambe le soluzioni si ottengono dalle funzioni di Bessel J, con l'espressione:

$$y = AJ_n(x) + BJ_{-n}(x)$$

dove A e B sono costanti da determinare con le condizioni al contorno.

Ordine	*Argomento*	*Funzioni*
1	1	0.4401
0	1	0.7652
1	0.5	0.2423
0	0.5	0.9385
1	10	0.04347
0	10	−0.2459
0.25	1	0.7522
0.25	1.5	0.6192
0.5	1.5708 ($\pi/2$)	0.6366 ($2/\pi$)
−0.25	1	0.6694
−0.25	1.5	0.3180
−0.75	1.5	−0.2684

Figura 11.14 — Alcuni valori della funzione di Bessel.

```
PROGRAM TSTBES(INPUT, OUTPUT);
(* prova della funzione di Bessel *)
(* è inclusa la funzione gamma *)

VAR
   DONE : BOOLEAN;
   X, ORDR : REAL;

FUNCTION GAMMA(X : REAL) : REAL;
 (* 26 Gen 81 *)

CONST
   PI = 3.1415926;

VAR
   I, J : INTEGER;
   Y, GAM : REAL;

BEGIN (* gamma function *)
   IF X >= 0.0 THEN
      BEGIN
         Y := X + 2;
         GAM := SQRT(2 * PI/Y) * EXP(Y * LN(Y)
                  + (1 - 1/(30 * Y * Y))/(12 * Y) - Y);
         GAMMA := GAM/(X * (X + 1))
      END
   ELSE (* x < 0 *)
      BEGIN
         J := 0;
         Y := X;
         REPEAT
            J :=J + 1;
            Y :=Y + 1.0
```

Figura 11.15 — La funzione di Bessel del primo tipo.

```
        UNTIL Y > 0.0;
        GAM := GAMMA(Y);
        FOR I := 0 TO J - 1 DO
            GAM := GAM/(X + I);
        GAMMA := GAM
    END (* IF *)
END (* gamma function *);

FUNCTION BESSJ(X, N : REAL) : REAL;
(* funzione di Bessel cilindrica *)
(* del primo tipo *)
(* è richiesta la funzione gamma *) )
(* 2 Feb 81 *)

CONST
  TOL = 1.0E-5;
  PI = 3.1415926;
VAR
  I : INTEGER;
  TERM, NEW_TERM, SUM, X2 : REAL;

BEGIN (* bessj *)
  X2 := X * X;
  IF (X = 0.0) AND (N = 1.0) THEN BESSJ := 0.0
  ELSE IF X > 15 THEN (* sviluppo asintotico *)
    BESSJ := SQRT(2/(PI * X)) * COS(X - PI/4 - N * PI/2)
  ELSE
    BEGIN (* serie infinita ordinata *)
      IF N = 0.0 THEN SUM := 1.0
      ELSE SUM := EXP(N * LN(X/2))/GAMMA(N + 1.0);
      NEW_TERM := SUM;
      I := 0;
```

Figura 11.15 — La funzione di Bessel del primo tipo (segue).

```
        REPEAT
          I := I + 1;
          TERM := NEW_TERM;
          NEW_TERM := -TERM * X2 * 0.25/(I * (N + I));
          SUM := SUM + NEW_TERM
        UNTIL ABS(NEW_TERM) <= ABS(SUM * TOL);
        BESSJ := SUM
      END (* IF *)
  END (* bessj *);

  BEGIN (* programma principale *)
    DONE := FALSE;
    REPEAT
      WRITE ('Ordine:');
      READLN (ORDR);
      IF ORDR < -25.0 THEN DONE := TRUE
      ELSE
        BEGIN
          WRITE(' X: ');
          READLN(X);
          WRITELN ('J Bessel è', BESSJ(X, ORDR))
        END
    UNTIL DONE
  END.
```

Figura 11.15 — La funzione di Bessel del primo tipo (segue).

PROGRAMMA PASCAL: FUNZIONE DI BESSEL DEL SECONDO TIPO

Se l'ordine dell'equazione di Bessel è intero, le soluzioni $J_n(x)$ e $J_{-n}(x)$ sono linearmente dipendenti; una seconda soluzione, indipendente, si può ottenere dall'espressione:

$$y = AJ_n(x) + BY_n(x)$$

dove Y è una funzione di Bessel del secondo tipo.

Il programma di Figura 11.18 può essere usato per calcolare le funzioni di Bessel del secondo tipo. L'ordine può essere qualsiasi numero reale, ma in genere queste funzioni si usano solo per ordini interi. Sono utilizzati due diversi algoritmi. Per argomenti di valore minore di 12, i valori di Y_0 e Y_1 sono calcolati con le espressioni:

$$Y_0(x) = \frac{2}{\pi} \sum_{m=0}^{n} (-1)^m \left(\frac{x}{2}\right)^{2m} \frac{[\ln \frac{x}{2} + \gamma - h]}{(m!)^2}$$

e

$$Y_1(x) = -\frac{2}{\pi x} + \frac{2}{\pi} \sum_{m=1}^{n} (-1)^{m+1} \left(\frac{x}{2}\right)^{2m-1} \frac{\left[\ln\left(\frac{x}{2}\right) + \gamma - h + \frac{1}{2m}\right]}{(m!)(m-1)!}$$

dove

$$h = \sum_{r=1}^{m} \frac{1}{r} \quad \text{se } m \geqslant 1$$

e Y è la costante di Eulero (Y = 0.57721566). Per ordini diversi da 0 e da 1 si usa la formula:

$$Y_n(x) = \frac{2n}{x} Y_{n-1}(x) - Y_{n-2}(x)$$

Per argomenti di valore maggiore si usa uno sviluppo asintotico simile a quello usato per le funzioni di Bessel J:

$$Y_n(x) = \sqrt{\frac{2}{\pi x}} \sin\left(x - \frac{\pi}{4} - \frac{n\pi}{2}\right)$$

La Figura 11.16 mostra un grafico delle funzioni Y_0 e Y_1.

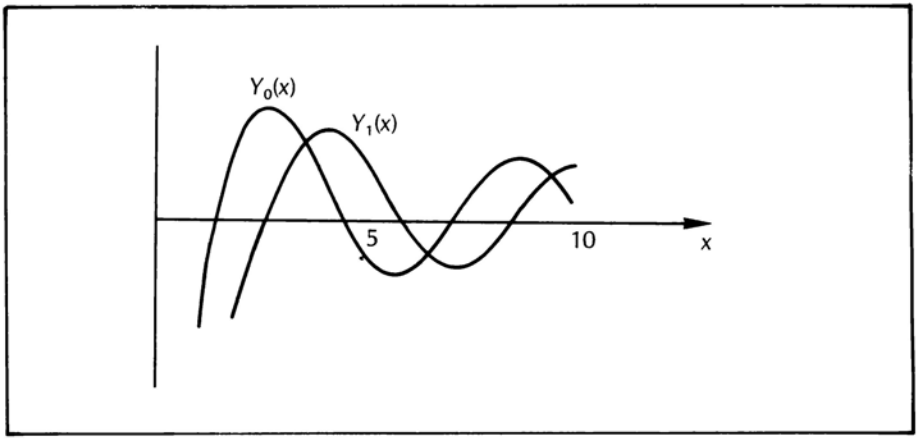


Figura 11.16 — Le funzioni di Bessel Y_0 e Y_1.

Scrivete il programma e provatelo. L'argomento deve essere un numero positivo, poichè la funzione va da $(-\infty)$ a 0. Nella Figura 11.17 sono mostrati alcuni valori tipici.

Argomento	Y_0	Y_1
1	0.088	−0.781
2	0.510	−0.107
3	0.377	0.325
11	−0.169	0.164
15	0.206	0.021

Figura 11.17 — Alcuni valori della funzione di Bessel del secondo tipo.

```
PROGRAM BESY(INPUT, OUTPUT);
(* valutazione della funzione di Bessel *)
(* del secondo tipo *)
(* 2 Feb 81 *)
VAR
   X, ORDR : REAL;
   DONE : BOOLEAN;
FUNCTION BESSY(X, N : REAL) : REAL;
(* funzione di Bessel cilindrica *)
(* del secondo tipo *)
(* 2 Feb 81 *)
CONST
   SMALL = 1.0E-8;
   EULER = 0.57721566;
   PI = 3.1415926;
   PI2 = 0.63661977 (* 2/pi *);
```

Figura 11.18 — La funzione di Bessel del secondo tipo.

```
VAR
   J : INTEGER;
   X2, SUM, SUM2, T, T2, TS, TERM, XX, Y0, Y1,
   YA, YB, YC, ANS, A, B, SINA, COSA : REAL;
BEGIN (* function bessy *)
  IF X < 12 THEN
    BEGIN
      XX := 0.5 * X;
      X2 := XX * XX;
      T := LN(XX) + EULER;
      SUM := 0.0;
      TERM := T;
      Y0 := T;
      J := 0;
      REPEAT
        J := J + 1;
        IF J <> 1 THEN SUM := SUM + 1/(J - 1);
        TS := T - SUM;
        TERM := -X2 * TERM/(J * J) * (1 - 1/(J * TS));
        Y0 := Y0 + TERM
      UNTIL ABS(TERM) < SMALL;
      TERM := XX * (T - 0.5);
      SUM := 0.0;
      Y1 := TERM;
      J := 1;
      REPEAT
        J := J + 1;
        SUM := SUM + 1/(J - 1);
        TS := T - SUM;
        TERM := (-X2 * TERM)/(J * (J - 1)) *
              ((TS - 0.5/J)/(TS + 0.5/(J - 1)));
        Y1 := Y1 + TERM
```

Figura 11.18 — La funzione di Bessel del secondo tipo (segue).

```
        UNTIL ABS(TERM) < SMALL;
        Y0 := PI2 * Y0;
        Y1 := PI2 * (Y1 - 1/X);
        IF N = 0.0 THEN ANS := Y0
        ELSE IF N = 1.0 THEN ANS := Y1
        ELSE

            BEGIN (* trova Y con la ricorsività *)
              TS := 2.0/X;
              YA := Y0;
              YB := Y1;
              FOR J := 2 TO TRUNC(N + 0.01) DO
                BEGIN
                  YC := TS * (J - 1) * YB - YA;
                  YA := YB;
                  YB := YC
                END;
              ANS := YC
            END;
          BESSY := ANS;
      END (* x < 12 *)
    ELSE (* x > 11, sviluppo asintotico *)
      BESSY := SQRT(2/(PI * X)) * SIN(X - PI/4 - N * PI/2)
END (* function bessy *);
BEGIN (* programma principale *)
  DONE := FALSE;
  WRITELN;
  REPEAT
    WRITE('Ordine?');
    READLN (ORDR);
    IF ORDR < 0.0 THEN DONE := TRUE
    ELSE
```

Figura 11.18 — La funzione di Bessel del secondo tipo (segue).

```
        BEGIN
          REPEAT
            WRITE(' Arg? ');
            READLN(X)
          UNTIL X >= 0.0;
          WRITELN('Y Bessel è', BESSY(X, ORDR))
        END (* IF *)
    UNTIL DONE
END.
```

Figura 11.18 — La funzione di Bessel del secondo tipo (segue).

Il programma ha termine se si dà un numero minore di zero.

SOMMARIO

Nel Capitolo 11 abbiamo rivisto alcuni concetti già affrontati in questo libro, e usato alcuni programmi scritti in precedenza. Con gli strumenti che abbiamo ora a disposizione siamo ormai in grado di realizzare alcune applicazioni matematiche veramente avanzate. Abbiamo studiato programmi in Pascal che calcolano con procedure diverse la funzione di errore Gaussiana, la funzione Gamma e le funzioni di Bessel. Sviluppando questi e altri programmi, abbiamo dato una dimostrazione dell'eleganza della programmazione in Pascal per applicazioni tecniche.

APPENDICE A

PAROLE RISERVATE E FUNZIONI DI SISTEMA

PAROLE RISERVATE

Le seguenti parole riservate del Pascal appaiono in neretto nel testo:

AND	**ARRAY**	**BEGIN**	**CASE**	**CONST**
DIV	**DO**	**DOWNTO**	**ELSE**	**END**
EXTERN*	**FILE**	**FOR**	**FUNCTION**	**GOTO**
IF	**IN**	**LABEL**	**MOD**	**NIL**
NOT	**OF**	**OR**	**PACKED**	**PROCEDURE**
PROGRAM	**RECORD**	**REPEAT**	**SET**	**THEN**
TO	**TYPE**	**UNTIL**	**VAR**	**WHILE**
WHITH				

* A seconda dell'implementazione

FUNZIONI DI SISTEMA

Nome	**Operazione eseguita**
ABS	Valore assoluto
ARCTAN	Arcotangente
CHR	Converte un numero intero in un carattere
COS	Coseno
EOLN	È vera quando si raggiunge la fine di una linea
EOF	È vera quando si raggiunge la fine di un file
EXP	e elevato a potenza
LN	Logaritmo naturale
ODD	È vera se l'argomento intero è dispari

ORD	Converte un carattere in un numero intero
PRED	Valore inferiore più vicino (di tipo carattere, intero o scalare)
ROUND	Converte un numero reale in un intero arrotondandolo
SIN	Seno
SQR	Elevamento al quadrato
SQRT	Radice quadrata
SUCC	Valore superiore più vicino (di tipo carattere, intero o scalare)
TRUNC	Converte un numero reale in intero troncandolo

APPENDICE B

SOMMARIO DEL LINGUAGGIO PASCAL

INSIEME MINIMO DEI CARATTERI STANDARD

Tutti i caratteri alfabetici	A – Z e/o a – z
Le cifre	0 – 9
I caratteri speciali	+ – * / = < > ( ) [ ]. , ; : ↑
Lo spazio (blank)	

NOMI DELLE VARIABILI

Il nome di una variabile in Pascal può essere una qualsiasi sequenza di caratteri alfabetici e numerici contigui. Alcuni compilatori accettano caratteri non alfanumerici, come il carattere della sottolineatura.

Esempi:

SEED LINE1 SUM_X2 Y_CALC

La lunghezza massima consentita per un nome varia da un compilatore all'altro; alcuni consentono un massimo di 6 o 8 caratteri, altri accettano fino a 32 caratteri e oltre. Perciò in questo libro sono stati usati identificatori come:

SUM_X_SQUARED

e

SUM_Y_SQUARED

invece di

SUM_SQUARE_X

e

SUM_SQUARE_Y

NUMERI

I numeri interi sono stringhe di cifre con o senza segno.

Esempi:

15 −19253 +7 0

I numeri reali sono scritti con il punto decimale o un fattore di scala, o entrambi:

379.1275 3.791275E2 3791275E−4

Il simbolo E indica la moltiplicazione per una potenza del 10 (E2 significa 10^2) e deve sempre essere seguito da un intero con o senza segno.

COMMENTI

I commenti sono scritti tra i simboli (* e *), e vengono ignorati dal compilatore.

Esempio:

(* riceve i valori per n e le matrici x, y *)

Alcuni compilatori accettano i simboli { e } come delimitatori di commenti.

OPERAZIONI

Operazioni intere

+	Addizione
−	Sottrazione
*	Moltiplicazione
DIV	Divisione
MOD	Modulo (fornisce il resto della divisione tra due interi)

Operazioni reali

+	Addizione
–	Sottrazione
*	Moltiplicazione
/	Divisione

Operazioni Booleane

:

AND OR NOT

Operazioni relazionali

Danno come risultato un valore Booleano VERO o FALSO (TRUE o FALSE).

<	Minore di
>	Maggiore di
=	Uguale a
<=	Minore o uguale di
>=	Maggiore o uguale di
< >	Diverso da
IN	Si usa con il tipo dati **SET** per determinare l'insieme d appartenenza di un elemento (Vedi "Tipi di insiemi").

SINTASSI

Intestazione del programma

PROGRAM PROGRAM_NAME (FILE_NAME,FILE_NAME, ...);

Esempio:

PROGRAM TSTSORT(INPUT,OUTPUT);

Definizione di costanti

CONST CONST_NAME = valore; CONST_NAME = valore;...

Esempio:

```
CONST
  MAXR = 20; (*punti*)
  MAXC = 4; (*termini polinomiali*)
```

In Pascal esistono alcune costanti predefinite:

TRUE	Valore Booleano vero
FALSE	Valore Booleano falso
MAXINT	Il più grande intero accettato dal calcolatore
NIL	Puntatore nullo

Notate che una costante deve essere definita con un unico valore, e non con un'espressione. La seconda delle istruzioni che seguono non è lecita:

```
CONST
  Y = 3; (legale)
  X = Y + 1 (illegale)
```

Definizione di variabili

VAR

VAR_NAME, VAR_NAME, ...: TYPE;
VAR_NAME, VAR_NAME, ...:. TYPE; ...

Esempio:

```
VAR
  X, X2: REAL;
  ALLDONE: BOOLEAN;
  ERROR: BOOLEAN;
```

Istruzioni di assegnamento

VAR_NAME: = espressione

Esempio:

```
X : = SEED + PI
X : = EXP(5.0 LN(X));
SEED : = X − TRUNC(X);
RANDOM : = SEED
```

Istruzioni composte

Un'istruzione composta è una sequenza di istruzioni comprese tra una **BEGIN** e una **END.**

Esempio:

```
BEGIN (*scambio*)
  HOLD : = P;
  P : = Q;
  Q : = HOLD
END (*scambio*)
```

Definizione di una procedura

PROCEDURE PROC_NAME (valore dei parametri; **VAR** parametri variabili);
corpo della procedura

Esempio:

```
PROCEDURE DERIV (X: REAL; VAR FX, DFX: REAL);

VAR
  E : REAL;


BEGIN (*deriv*)
  E : = EXP(X);
  FX : = E − 4.0 * X;
  DFX : = E − 4.0
END (*deriv*)
```

Definizione di funzioni

FUNCTION FUNC_NAME (parametri): tipo_risultato; corpo della funzione. Il valore del risultato della funzione viene assegnato (nel corpo della funzione) con un'istruzione del tipo

```
FUNCT_NAME: = valore;
```

Esempio:

```
FUNCTION DETER (A : ARY 2) : REAL;
(* Calcola il determinante di una matrice 3-per-3 *)
VAR
  SUM : REAL;
BEGIN (*deter*)
  SUM : = A[1,1] * (A[2,2] * A[3,2] - A[3,2]  * A[2,3])
          - A[1,2] * (A[2,1] * A[3,3] - A[3,1] * A[2,3])
          + A[1,3] * (A[2,1] * A[3,2] - A[3,1] * A[2,2]);
  DETER : = SUM
END (*deter*);
```

Uso del punto e virgola (;)

Fra due istruzioni Pascal è sempre necessario un punto e virgola; non è necessario immediatamente dopo un **BEGIN** o immediatamente prima di una **END**, ma è innocuo. Non deve mai precedere immediatamente la **ELSE** nella costruzione **IF-THEN-ELSE.**

ISTRUZIONI CONDIZIONALI

L'istruzione IF-THEN

IF espressione **THEN** istruzione

Esempio:

```
IF B [N,N] = 0.0 THEN ERROR: = TRUE
```

L'istruzione IF - THEN - ELSE

```
IF espressione THEN istruzione ELSE istruzione
```

Esempio:

```
IF DET = 0.0 THEN
  BEGIN
    ERROR : = TRUE;
    WRITELN ('ERRORE: matrice singolare')
  END
ELSE
  BEGIN
    SETUP(B,COEF,1);
    SETUP(B,COEF,2);
    SETUP(B,COEF,3)
END (*ELSE*)
```

L'istruzione CASE

```
CASE espressione OF
  etichette: istruzione;
  etichette: istruzione;

                .

  etichette: istruzione;
END
```

Esempio:

```
CASE YEAR OF
  1: WRITE('FRESHMAN');
  2: WRITE('SOPHOMORE');
  3: WRITE('JUNIOR');
  4: WRITE('SENIOR');
5,6,7: WRITE('GRADUATE')
END;
```

ISTRUZIONI ITERATIVE

L'istruzione WHILE - DO

WHILE espressione **DO** istruzione

Esempio:

```
WHILE PIVOT < A[J]DO J : = J − 1;
```

L'istruzione REPEAT - UNTIL

REPEAT gruppo di istruzioni **UNTIL** espressione

Esempio:

```
REPEAT
  WRITE('Quante equazioni?');
  READLN(N)
UNTIL N<MAXR;
```

L'istruzione FOR - TO - DO

FOR variabile : = primo valore **TO** ultimo valore **DO** istruzione

Esempio:

```
FOR J : = 1 TO N DO WRITE (COEF[J]);
```

L'istruzione FOR - DOWNTO - DO

FOR variabile : = primo valore **DOWNTO** ultimo valore **DO** istruzione

Esempio:

```
FOR I : = TERMS DOWNTO 1 DO
  BEGIN
    SUM : = 1.0 + I * V/U
    U : = SUM
  END;
```

ISTRUZIONI DI TRASFERIMENTO DEL CONTROLLO

Vi raccomandiamo di evitare le istruzioni GOTO tranne in quei rari casi in cui hanno effettivamente un solo valore.

Dichiarazione di etichetta (label)

Se in un programma Pascal vengon usate delle label, devono essere dichiarate esplicitamente in precedenza, immediatamente dopo la sezione di dichiarazione delle costanti CONST.

```
LABEL intero, intero, ...;
```

Esempio:

```
LABEL 99;
```

L'istruzione GOTO

```
GOTO label
```

Esempio:

```
BEGIN (*Procedure gaussj*)
  IF ERROR THEN GOTO 99;



  99:
END (*Procedure gaussj*);
```

In questo esempio, l'istruzione **GOTO** causa la fine sistematica della procedura se ERROR è vera. L'istruzione alternativa, **IF NOT** ERROR **THEN**, sarebbe più complicata da gestire. L'uso dell'istruzione **GOTO** per uscire da una procedura è una tecnica di programmazione discutibile, anche se alcuni compilatori l'accettano.

INGRESSO E USCITA

Procedure di Ingresso

```
READ (variabili);
READLN (variabili);
```

```
READ (FILE_NAME), variabili);
READLN (FILE_NAME, variabili);
```

La variabile FILE_NAME deve essere esplicitamente dichiarata di tipo **FILE**, o deve essere passata come parametro nell'intestazione del programma. Notate che il tipo TEXT è equivalente a **FILE OF CHAR**.

L'istruzione READ attribuisce i valori della linea corrente ("record") alle sue variabili, mentre i valori restanti sono disponibili per una successiva istruzione di ingresso.

L'istruzione READLN assegna alle sue variabili i valori della linea corrente, ignorando quelli restanti. Una successiva istruzione di ingresso preleva i valori dall'inizio della linea seguente.

Procedure di Uscita

```
WRITE (espressioni);
WRITELN (espressioni);
WRITE (FILE_NAME, espressioni);
WRITELN (FILE_NAME, espressioni);
```

L'istruzione WRITE produce un'uscita sulla linea corrente ("record"). L'istruzione WRITELN produce un'uscita sulla linea corrente, e forza un "ritorno a capo" per terminare il record.

Esempi:

```
WRITE('X:');
READLN(X);
WRITELN('GAMMA È', GAMMA (X));
```

Formato di un'uscita numerica

Ogni espressione in un'istruzione WRITE o WRITELN può essere seguita da uno o due interi nel modo seguente:

```
intero_espressione: intero
reale_espressione: intero: intero
```

In uscita viene fornito il valore dell'espressione. Il primo intero rappresenta l'ampiezza in caratteri del campo dell'uscita; il valore effettivo dell'uscita è posto sulla destra del campo, preceduto eventualmente da blank. In uscita compaiono sempre tutte le cifre del valore, anche se superano l'ampiezza del campo.

Nel caso di un numero reale, il secondo intero definisce il numero di cifre che si devono trovare a destra del punto decimale.

Esempio:

```
WRITELN(I:3, X[I]:8:1, Y[I]:9:2, Y_CALC[I]:9:2);
```

TIPI DI DATI

La dichiarazione **TYPE** segue la **CONST** e precede la **VAR**.

Tipo scalare

```
TYPE TYPE_NAME = (identificatore, identificatore, ...);
```

Esempio:

```
TYPE
  DAY_OF_WEEK = (MONDAY, TUESDAY, WEDNESDAY,
                 THURSDAY, FRIDAY,
                 SATURDAY, SUNDAY);
```

Tipo subrange

```
TYPE
  TYPE_NAME = costante..costante;
VAR
  VAR_NAMES: costante..costante;
```

Esempio:

```
TYPE
  INDEX = 1..MAX;
  WEEK_DAY = (MONDAY..FRIDAY);
```

Tipo matrice

```
TYPE
  TYPE_NAME = ARRAY [tipo_indice, tipo_indice,...]
              OF tipo;
```

```
VAR
  VAR_NAMES : ARRAY [tipo_indice, tipo_indice,...]
              OF tipo;
```

Una matrice multidimensionale richiede che sia specificato un tipo_indice per ogni dimensione. Il tipo_indice è di solito un tipo subrange, ma può essere un tipo scalare.

Esempi:

```
TIPE
  ARYS  = ARRAY[1..CMAX]OF REAL;
  ARY2S = ARRAY[1..RMAX, 1..CMAX] OF REAL;
          (* Bidimensionale *)
VAR
  W     : ARRAY[1..MAXC, 1..MAXC] OF REAL;
        : (* MAXC PER MAXC *)
  INDEX : ARRAY[1..MAXC, 1..3] OF INTEGER;
          (MAXC PER 3)
```

Riferimenti agli elementi di una matrice

Gli indici di una matrice compaiono tra parentesi quadre. Nel caso di matrici multidimensionali, le espressioni degli indici sono separate da virgole. Un indice può essere qualsiasi espressione il cui risultato sia conforme al tipo dichiarato.

```
ARRAY_NAME [indice, indice,...]
```

Esempio:

```
C[I,J] : = A[I,K] * B[K,J]
```

Matrici "packed"

In alcune implementazioni, la quantità di memoria occupata da una matrice può essere ridotta dichiarando la matrice **PACKED**. Lo spazio effettivamente risparmiato, e il conseguente aumento del tempo di accesso dipendono dall'implementazione.

```
TYPE
  TYPE_NAME = PACKED ARRAY [tipo_indice, tipo_indice...]
              OF tipo_elemento;
```

```
VAR
  VAR_NAME: PACKED ARRAY [tipo_indice, tipo_indice...]
                OF tipo_elemento;
```

Esempio:

```
TYPE
  LINE_IMAGE = PACKED ARRAY [1...MAXCOL] OF CHAR
```

Tipi record

```
TYPE
  TYPE_NAME = RECORD lista campi END;
VAR
  VAR_NAME: RECORD lista campi END;
```

Esempi:

```
TYPE
  COMPLEX = RECORD RE, IM: REAL END;
TYPE
  CUSTOMER = RECORD
    NAME,STREET: PACKED ARRAY[1..30] OF CHAR:
    CITY:PACKED ARRAY[1..20] OF CHAR:
    STATE:PACKED ARRAY[1..2] OF CHAR;
    ZIP:0..99999
  END(*record cliente*);
```

I campi di record variabili possono cambiare da un record a un altro; le diverse strutture che si possono incontrare nella parte variabile del record sono specificate con una costruzione del tipo **CASE**, che deve seguire la dichiarazione dei campi fissi dei record.

Esempio:

```
TYPE
  DATE = RECORD
    MO: 1..12;
    YR:INTEGER
  END(*data*);
  GRADE = (FRESH, SOPH, JR, SEN);
```

```
STUDENT = RECORD
  NAME:PACKED ARRAY [1..20] OF CHAR;
  NUM: INTEGER;
  CASE GR:GRADE OF
    FRESH, SOPH: (MAJOR: BOOLEAN);
    JUN: (CREDITS: INTEGER);
    SEN: (GDATE: DATE)
END(*studente*)
```

Riferimenti a campi di record

Un campo all'interno di un record variabile è identificato dalla cosiddetta notazione "dot"; il nome del campo, che appare nella dichiarazione del tipo record, è congiunto, tramite un punto (dot), all'effettivo nome del record:

```
RECORD_NAME.FIELD_NAME
```

Esempi:

```
VAR
  COED: STUDENT; (*Vedi tipo STUDENT precedente*)
  ...
COED.NAME : = 'JANE SMITH      ';
COED.NUM : = 75210;
COED.GR : = SEN;
COED.GDATE.MO : = 6;
COED.GDATE.YR : = 1981;
```

Quando si deve accedere molte volte a dei campi di uno stesso record, si può usare l'istruzione **WITH** nel modo seguente:

```
WITH RECORD_NAME DO istruzione
```

Esempio:

```
VAR
  CUR_ACCOUNT:CUSTOMER; (* Vedi tipo CUSTOMER precedente *)
...
WITH CUR_ACCOUNT DO
  BEGIN
    READLN(NAME);
    READLN(STREET);
    READLN(CITY,STATE,ZIP)
  END;
```

La costruzione **WITH** rende il programma più chiaro da leggere e più facile da compilare.

Tipi insiemi

```
TYPE
  TYPE_NAME = SET OF tipo_base;
VAR
  VAR_NAME : SET OF tipo_base;
```

Esempio:

```
TYPE
  DAYS = (MO, TU, WE, TH, FR, SA, SU);
  WEEK = SET OF DAYS;
VAR
  WORKDAY, HOLIDAY, WEEKDAY: WEEK;
```

Operazioni su insiemi

+	Unione
*	Intersezione
–	Differenza
=	Equivalenza
<>	Non equivalenza
IN	Appartenenza all'insieme; è TRUE se l'operando scalare a sinistra è un elemento dell'insieme sulla destra.
<=	Inclusione; è TRUE se tutti gli elementi **IN** operando a sinistra sono anche **IN** quello di destra.
>=	Contenimento: è TRUE se tutti gli elementi **IN** operando a destra sono anche **IN** quello a sinistra.

Tipi File

```
TYPE
  nome-file = FILE OF tipo;
VAR
  lista-variabili: FILE OF tipo;
```

Esempio:

```
VAR
  FILEA, FILEB, FILEC : FILE OF INTEGER;
```

Le procedure standard di Ingresso/Uscita (Input/Output, I/O) sui file sono:

RESET	Apre un file per operazioni di lettura
REWRITE	Apre un file per operazioni di scrittura (tutti i dati scritti in precedenza sul file sono persi).
GET	Trasferisce un elemento di un file nel buffer associato.
PUT	Trasferisce il contenuto di un buffer nel file corrispondente.
EOF	Funzione Booleana che è TRUE quando si è raggiunta la fine di un file.

Tipi puntatori

Un puntatore contiene l'indirizzo di una variabile dinamica, cioè di una variabile creata durante l'esecuzione di un programma con la procedura NEW. Queste variabili sono identificate indirettamente tramite dei puntatori. Un puntatore può essere inizializzato con il valore **NIL**, per indicare che non sta puntando nessuna variabile dinamica.

```
TYPE
  nome_tipo = tipo;
VAR
  nome_variabile = tipo;
```

Esempio: (Lista concatenata)

```
TYPE
  LINK = PART: (* LINK È UN PUNTATORE AD UNA VOCE DEL TIPO
                  PART*)
  ...
  PART = RECORD
  ...
    NEXT:LINK;(*PUNTA AL PROSSIMO PART*)
    ...
  END (*PART RECORD*);
```

PROGRAMMI DI MATEMATICA E STATISTICA

Leggendo questo libro il lettore potrà formarsi quella logica di base indispensabile per la risoluzione di problemi di matematica e statistica.
Ad ogni programma viene preposta un'esposizione schematica del metodo numerico e delle tecniche di programmazione utilizzate, il diagramma a blocchi relativo all'algoritmo, il listato (anch'esso ottenuto da calcolatore) in cui tra l'altro vengono specificati il tempo e la quantità di memoria impiegati.

Cod. 522D **L. 16.000 Pagg. 228**

INTRODUZIONE AL PASCAL

Il volume, incentrato su numerosissimi esempi che verificano costantemente l'apprendimento del lettore, insegna a conoscere, capire ed usare tutte le particolarità e i vantaggi di questo linguaggio. Nel corso della trattazione vengono ampiamente utilizzate le tecniche di programmazione strutturata, come pure tecniche particolari, quali il trattamento dei file, l'utilizzazione della recursività e il trattamento grafico.

Cod. 516A **L. 30.000 Pagg. 484**

COMPUTER GRAFICA

Si può dire che la computer grafica si pone nel contesto più generale del trattamento dell'informazione, avendo individuato nell'immagine un contenuto informativo che è possibile elaborare.
Quest'opera, con il suo rigore informativo e scientifico, si pone come fondamentale nel carente panorama italiano; inoltre le informazioni e gli spunti contenuti nel testo contribuiranno certamente alla divulgazione ed alla formazione di idee nuove e feconde.

Cod. 519P **L. 29.000 Pagg. 174**

CEDOLA DI COMMISSIONE LIBRARIA

Ritagliare (o fotocopiare) e inviare a
Gruppo Editoriale Jackson Via Rosellini, 12 - 20124 Milano

Nome e Cognome _____

_____

Indirizzo _____

_____

Cap. _____ Città _____ Provincia _____

Codice Fiscale (indispensabile per le aziende)

Inviatemi i seguenti libri:

Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità

☐ Pagherò al postino il prezzo indicato + L. 2.000 per contributo fisso spese di spedizione

☐ Allego assegno n° ..... di L. .....

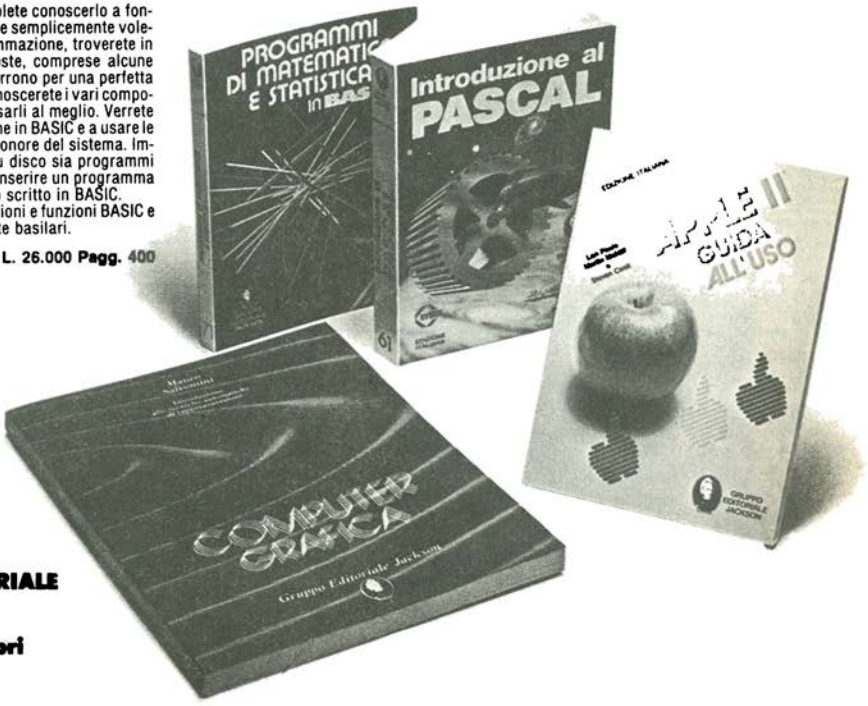
Data ..... Firma .....

... dalla libreria JACKSON

APPLE II - Guida all'uso

Se possedete un Apple e volete conoscerlo a fondo, se volete comprarlo, o se semplicemente volete imparare la sua programmazione, troverete in questo libro, tutte le risposte, comprese alcune vere "primizie" che vi occorrono per una perfetta operatività del sistema. Conoscerete i vari componenti del sistema e come usarli al meglio. Verrete guidati alla programmazione in BASIC e a usare le caratteristiche grafiche e sonore del sistema. Imparerete a memorizzare su disco sia programmi che archivi dati, come ad inserire un programma scritto in assembler in uno scritto in BASIC.
E poi ancora, tutte le istruzioni e funzioni BASIC e ben 12 appendici veramente basilari.

Cod. 331P **L. 26.000 Pagg. 400**



GRUPPO EDITORIALE JACKSON

Divisione Libri

L'ESIGENZA DI CAPIRE

oscillante, ordinamento
oscillatore *m*
oscillatore a battimenti
oscillatore a quarzo
oscillatore a rilassamento
oscillatore di rilassamento a bloccaggio
oscillatore quarzato
oscillazione /
oscillazione / a campo ritardante
oscillazione pendolare
oscillografo *m*
oscillografo a raggi catodici
oscillogramma *m*
oscilloscopio *m*
ospite, calcolatore ~
ottale *agg*
ottale, cifra
ottetto *m* (byte composto da 8 bit)
ottica delle fibre
ottico *agg*
ottimare ‹

oscillating sort
oscillator *n*
beat frequency oscillator
quartz oscillator
astable multivibrator
blocking oscillator
quartz oscillator
oscillation *n*
retarding-field oscillation
hunting *n* (unstable condition)
oscillograph *n*
cathode-ray oscillograph

8-bit byte
fiber optics
optically readable
optimize ‹

metodo

metodo di accesso sequenziale
metodo di lettura
metodo di Monte Carlo
metodo di ordinamento
metodo di programmazione
metodo di registrazione
metodo di ricerca
metodo di riconoscimento errori
metodo di trasmissione
mettere a punto (un programma ecc)
mettere a punto in linea
mezza parola
mezzi *m pl*
mezzo *m*
mezzo addizionatore
mezzo di immagazzinamento
mezzo di protezione dati
mezzo fisico di trasmissione
mezzo trasmissivo (hardware per il controllo della trasmissione)
MF = modulazione di frequenza
micro *m*
microcalcolatore *m* (calcolatore costruito intorno ad un microprocessore, generalmente completo di memoria e periferiche)
microcalcolatore, sistema a ~
microcalcolatore monochip
microcalcolatore su singola scheda
microciclo *m*
microcircuito *m*
microcircuito integrato
microcomputer *m*
microcomputer, piastra di ~
microcomputer didattico
microcomputer single chip
microcomputer su un unico chip
microcontroller *m*
microelaboratore *m* (raro) v. microprocessore
microelettronica /
microfiche /
microfilm *m*
microfilm, uscita su ~
microfilmatura /
microfotogramma *m*
microistruzione /
microistruzione /
micrologica /
micrologico *m*
micrologico LSI
microminiaturizzato, circuito ~
microminiaturizzazione /
micromodulo *m*
microonde /
microoperazione /
microplacchetta /
microplacchetta *f* (di un microcomputer)
microprocessore (µP) *m*
microprocessore a chip singolo

sequential access method (SAM)
reading method
Monte-Carlo method
sorting method
programming ‹

trasmettere

trasmettere ‹
trasmettere ‹
trasmettere per telescrivente
trasmettitore *m*
trasmettitore *m*

transmit ‹
send ‹
teleprint ‹
transmitter *n*
sender *n*

initialize ‹ (tape etc)
initialize ‹ (program)
initialize, to ~ a volume

conversion

conversion *n* (from old system to new one etc)
conversion (of signals etc)
conversion, program ~
conversion instruction
conversion routine (see translation program)
conversion table, translation table

conversion time
conversion transconductance
conversion unit, converter *n*

convert ‹ (data etc)
converter *n* see conversion unit, converter *n*

inizializzare ‹ (nastro ecc)
inizializzare ‹ (programma)
inizializzare un volume

trasformazione *f* (di un impianto ecc)
trasformazione *f*, conversione *f*
conversione di programmi
istruzione di conversione
programma di conversione
tabella di conversione

assestamento
trascondutanza di conversione
convertitore *m*, convertitrice *f*

converture ‹ (dati, etc)

vorbereiten v, etikettieren v
vorbereiten ‹
einen Datenträger einrichten, einen

Umrüstung *f*, Umstellung *f*
Umformung *f*, Umsetzung *f*
Programmkonversierung *f*
Umsetzbefehl *m*
Umrüstprogramm *n*
Umsetztabelle *f*, Umsetzungstabelle *f*
Umwandlungszeit *f*
Mischsteilheit *f*
Umsetzer* *m*, Umwandler *m*, Konverter *m*
umsetzen ‹, umwandeln ‹

Stromrichter *m*
Parallel-Serien-Umsetzer (PSU)* *m*
Serien-Parallel-Umsetzer *m*

Kühlluftgebläse *n*

Koordinaten *f pl*
Koordinierung *f*

Kopiergerät *n*

kopieren ‹, duplizieren v

Kopie *f*, Duplikat *n*, Zweitexemplar *n*
Kopischlag *m* (Kopie)
Kopierfachungs-Funktion *f*
Kopierhalter *m*
Kopierin *n*
Kopierer(elektr.)
Kopiersystem *n*

Kernspeichermatrix *f*

Steuerung *f*

Bibliothek

Antivalenzglied *n*
Antrieb *m*
Antriebskette *f*
Antriebsmotor *m*
Antriebsregelung *f* (der Magnetbandeinheit)
Antriebswelle *f*
Antwort *f*
Antwort des Operators
antworten ‹
Antwortmeldung *f*
Antwortsignal *n*
Antwortzeit *f*
Anweisung* *f*
Anweisung *f*
Anweisung *f* (zur Gerätebedienung etc)
Anweisung *f* (bei problemorientierten Programmiersprachen)
Anweisung *f* (bei maschinenorientierten Sprachen)
Anweisung an das Programm
Anweisung in Primärsprache
Anweisung, arithmetische ~
Anweisung, ausführbare ~
Anweisung, nichtausführbare ~
Anweisung, symbolische ~
Anweisung, unbedingte ~
Anweisung, zusammengesetzte
Anweisungsmarke *f*
Anweisungsnummer *f*
Anwender *m*
Anwenderarbeitsbereich *m*
Anwenderdatei *f*
Anwenderebene *f*
Anwendermakro *m*
Anwendermaske *f*
anwenderorientiert *adj*
Anwenderprogramm *n*
Anwenderprogrammierung *f*
Anwenderschnittstelle *f*
Anwendersoftware *f*
Anwendersoftware *f* (vom Anwender geschrieben)
Anwendersysteme, dedizierte ~
Anwenderunterprogramm *n*

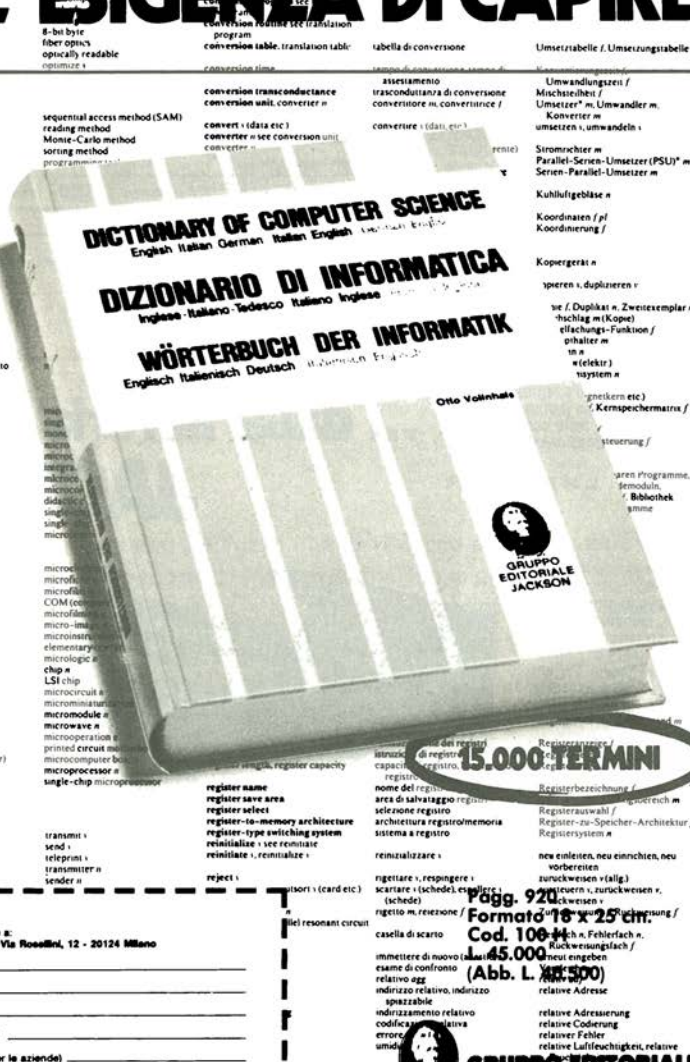
sequentielle

sequentielle Steuerung
sequentielle Struktur
sequentielle Verarbeitung
sequentielle Zugriffsmethode
sequentieller Rechner
sequentieller Speicher
sequentieller Zugriff
Sequentiellrechner *m*
Sequenz *f* (Sätze oder Zeichen)
Sequenzspeicher *m*
Serialdrucker *m*
Serialdruckwerk* *n*
Serie *f*
Serie *f*
Serie, in ~
seriell *adj* (Übertragung etc)
seriell *adj*
seriell aufgebaut
serielle asynchrone Schnittstelle
serielle Binärübertragung
serielle Organisation
serielle Start-Stopp-Übertragung
serielle Synchronübertragung
serielle Übertragung
serielle Verarbeitung
serieller Betrieb*
serieller Datentransfer
serieller Ein-/Ausgabekanal
serieller I/O-Port
serieller Port
serieller Zugriff
Serienabtastung *f*
Serienaddition *f*
Serienausgang *m*
Serienbetrieb *m*
Serendrucker *m*
Serieneingang *m*
Serienmultiplikation *f*
Serennummer *f*
Serien-Parallelbetrieb *m*
Serien-Parallel-Schaltung *f*
Serien-Parallelsystem *n*
Serien-Parallel-Umsetzer *m*
Serien-Parallel-Umsetzung *f*
Serienprogrammierung *f*
Serienrechner *m*
Serienschaltung *f*
Serienschnittstelle *f*
Serien-Serien-Betrieb *m*
Serienspeicher *m*
Serienspeicherung *f*
Serientransfer *m*
Serienübertragung *f*
Serienverarbeitung *f*
Service-Rechenzentrum *n*
Service-Techniker *m*
Servomechanismus *m*
Servomotor *m*
Servosystem *n*
Set *n*

grafische

grafische Einheit (normierte Koordinaten)
grafische Lösung
grafisches Ausgabegerät
grafisches Datenverarbeitungssystem
grafisches Symbol
grafisches Terminal
Grammatik *f*
grammatische Analyse
grammatischer Fehler
Graph *m*
Graphik... ‹ Grafik...
graphisch *adj* ‹ grafisch
Gray-Code *m*
Grenzfrequenz *f*

Grenzpriorität *f*



register name
register save area
register select
register-to-memory architecture
register-type switching system
reinitialize ‹ see reinitiate
reinitiate ‹, reinitialize ‹

reject ‹

nome del registro
area di salvataggio registro
selezione registro
architettura registro/memoria
sistema a registro

reinizializzare ‹

rigettare ‹, respingere ‹
scartare ‹ (schede), espellere ‹ (schede)
rigetto *m*, reiezione *f*

casella di scarto

immettere di nuovo (schede)
esame di confronto
relativo *agg*
indirizzo relativo, indirizzo spiazzabile
indirizzamento relativo
codifica relativa
errore relativo
umidità relativa

programmazione relativa, programmazione in relativo

relè *m*, relais *m*
calcolatore a relais
campo di accoppiamento relè
liberare ‹, abilitare ‹ (unità ecc.), rilasciare ‹ (buffer ecc.)
liberazione *f*, rilascio *m* (di unità ecc.)

affidabilità *f*

affidabilità valutata
affidabilità dell'hardware
affidabilità ottimale, sicurezza di esercizio ottimale

Registeranzeiger /
Registerbezeichnung /
Registerbereich *m*
Registerauswahl /
Register-zu-Speicher-Architektur *f*
Registersystem *n*

neu einleiten, neu einrichten, neu vorbereiten
zurückweisen v (allg.)
zurückweisen v, zurückweisen v, zurückweisen ‹
Rückweisung /
Rückweisungsfach /
neu eingeben
Fehlerfach *n*, Fehlerfach *n*,
Rückweisungsfach /
vorbereiten
relative Adresse

relative Adressierung
relative Codierung
relativer Fehler
relative Luftfeuchtigkeit, relative Feuchtigkeit
relative Programmierung

Relais* *n*
Relaisrechner *m*
Relaiskoppelfeld *n*
freigeben ‹ (Gerät, Puffer etc.)

Freigabe *f*
Zuverlässigkeit *f*, Betriebssicherheit *f*
geschätzte Zuverlässigkeit
Hardware-Zuverlässigkeit *f*
optimale Betriebssicherheit

15.000 TERMINI

Pagg. 920
Formato 16 x 25 cm.
Cod. 100
L. 45.000
(Abb. L. 35.500)

GRUPPO EDITORIALE JACKSON

Tagliando d'ordine da inviare a:
Gruppo Editoriale Jackson - Via Rosellini, 12 - 20124 Milano

NOME _____

COGNOME _____

INDIRIZZO _____

CAP _____ CITTA' _____

Partita IVA (indispensabile per le aziende) _____

☐ Inviatemi il **Dizionario di Informatica.**

☐ Pagherò al postino l'importo indicato più L. 2.000 per le spese di spedizione.

☐ Allego assegno n° _____ di L. _____

☐ Non Abbonato

☐ Abbonato (sconto 10%): _____
(indicare a quali riviste)

DATA _____ FIRMA _____

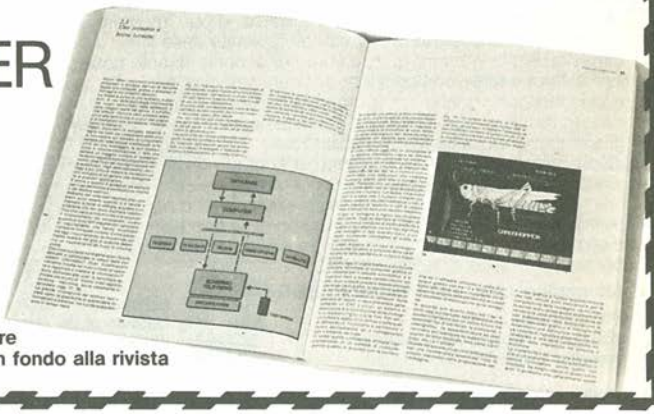
Quando il computer parla il linguaggio delle immagini

La computer grafica rappresenta un campo di applicazione dell'informatica relativamente nuovo, ma suscettibile di imprevedibili sviluppi. Questo volume, nato in collaborazione con alcune delle più specializzate istituzioni del settore, esamina tutte le possibilità di questa scienza nuova e affascinante: dall'animazione cinematografica e televisiva ai businnes graphics; dalla progettazione in architettura a quella in elettronica e in meccanica; dalla mappazione alla manipolazione tridimensionale delle immagini... Realizzata in modo da permettere un rapido, ma esauriente approccio all'argomento, l'opera si rivolge a quanti (lettori-utenti) siano alla ricerca dei necessari chiarimenti per una corretta e proficua utilizzazione delle tecniche di Computer grafica.

Mauro Salvemini

COMPUTER GRAFICA

176 pagine. Lire 29.000
Codice 519 P



GRUPPO EDITORIALE JACKSON

Per ordinare il volume utilizzare l'apposito tagliando inserito in fondo alla rivista

È vero: piccolo è bello!

Alla scoperta dello ZX SPECTRUM

a cura di **Rita Bonelli**

ZX Spectrum è l'ultimo nato della famiglia Sinclair. È un calcolatore a colori di piccole dimensioni, ma di grandissime possibilità. Imparare a usarlo bene può essere fonte di molte piacevoli scoperte. Questo libro vi aiuta a raggiungere lo scopo. In 35 brevi e facilissimi capitoli non solo imparerete tutto sulla programmazione in BASIC, ma arriverete anche a usare efficientemente il registratore e a sfruttare al meglio le stampe. Soprattutto capirete la differenza tra il vostro Spectrum e gli altri computer.

320 pagine. Lire 22.000 Codice 337 B

GRUPPO EDITORIALE JACKSON

Per ordinare il volume utilizzare l'apposito tagliando inserito in fondo alla rivista

Scrive, suona, gioca, entusiasma

Gaetano Marano

66 PROGRAMMI PER ZX 81

E ZX 80 CON NUOVA ROM + HARDWARE

Per le sue qualità e il suo modestissimo prezzo lo ZX 81 della Sinclair è il computer più venduto nel mondo.
Oggi, sempre con una modestissima spesa, si può imparare a sfruttare questo eccezionale strumento al limite delle sue capacità. Basta scorrere questo libro per scoprire quante cose lo ZX 81 può fare con l'aggiunta di alcuni semplici ed economici componenti. Ad esempio, tramite un semplice circuito musicale può riprodurre 50 note su 4 ottave e, sempre grazie a una modifica hardware da poche migliaia di lire, lo ZX 81 diventa anche l'unico computer in grado di conferire effetti sonori ai giochi inseriti tra i suoi programmi. Ma non è tutto. Un'altra novità di quest'opera, preziosa anche per chi possiede lo ZX 80 con ROM, è il regalo di alcune tastiere disegnate da sovrapporre a quella sensitiva dell'apparecchio, per ricavarne altre, speciali funzioni.

136 pagine. Lire 12.000 Codice 520 D

Per ordinare il volume utilizzare l'apposito tagliando inserito in fondo alla rivista

GRUPPO EDITORIALE JACKSON

Una guida pratica, preziosa, aggiornata

General Electric

LA SOPPRESSIONE DEI TRANSITORI DI TENSIONE

Un libro che riassume i risultati delle pluriennali ricerche effettuate da una delle massime industrie mondiali sulle cause, gli effetti, la frequenza dei sovraccarichi di tensione derivanti dai disturbi atmosferici o da altri motivi. Un'opera eminentemente pratica che si propone di dare ai tecnici un contributo fattivo alla soluzione di questo annoso problema, anche attraverso l'indicazione della vasta gamma di dispositivi di protezione che la G.E. ha messo a punto sulla scorta dei suoi studi e delle esperienze.

216 pagine. Lire 12.000 Codice 611 A

GRUPPO EDITORIALE JACKSON

Per ordinare il volume utilizzare l'apposito tagliando inserito in fondo alla rivista

CEDOLA DI COMMISSIONE LIBRARIA

Da inviare a Gruppo Editoriale Jackson - Via Rosellini, 12 - 20124 Milano

Nome Cognome

Indirizzo

Cap. **Città** **Provincia**

Partita I.V.A. (indispensabile per le aziende)

Si richiede l'emissione della fattura ☐

Inviatemi i seguenti libri:

Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità

☐ Pagherò al postino il prezzo indicato + L. 2.000 per contributo fisso spese di spedizione.

☐ Allego assegno n° ..... di L. .....

☐ Non abbonato ☐ Abbonato sconto 20% ☐ l'Elettronica ☐ Elettronica Oggi ☐ Automazione Oggi ☐ Elektor
☐ Informatica Oggi ☐ Computerworld ☐ Bit ☐ Personal Software ☐ Strumenti Musicali ☐ Videogiochi

Data ..... Firma .....

CEDOLA DI COMMISSIONE LIBRARIA

Da inviare a Gruppo Editoriale Jackson - Via Rosellini, 12 - 20124 Milano

Nome Cognome

Indirizzo

Cap. **Città** **Provincia**

Partita I.V.A. (indispensabile per le aziende)

Si richiede l'emissione della fattura ☐

Inviatemi i seguenti libri:

Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità	Codice Libro	Quantità

☐ Pagherò al postino il prezzo indicato + L. 2.000 per contributo fisso spese di spedizione.

☐ Allego assegno n° ..... di L. .....

☐ Non abbonato ☐ Abbonato sconto 20% ☐ l'Elettronica ☐ Elettronica Oggi ☐ Automazione Oggi ☐ Elektor
☐ Informatica Oggi ☐ Computerworld ☐ Bit ☐ Personal Software ☐ Strumenti Musicali ☐ Videogiochi

Data ..... Firma .....

L. 25.000 Cod. 554P ISBN88-7056-134-8

L'obiettivo di questo libro è duplice: aiutare il lettore a perfezionarsi nell'uso del Pascal, e insieme costruire una libreria di programmi per risolvere i problemi, che si incontrano più frequentemente nel campo scientifico e ingegneristico.
Gli algoritmi matematici di ogni programma sono descritti sistematicamente prima di implementare il programma stesso. Per quasi tutti i programmi viene fornito un esempio dei risultati prodotti. Tutti i programmi in PASCAL sono stati sviluppati su un microcalcolatore Z-80 con sistema operativo CP/M.
Talvolta vengono usate costruzioni comuni ad altri linguaggi evoluti (come il FORTRAN, il BASIC e l'ALGOL), al posto di quelle più eleganti del PASCAL: ad esempio, si usano preferibilmente le matrici al posto dei record. Perciò gli algoritmi presentati nel testo, possono essere facilmente convertiti in altri linguaggi.
Per chi affronta il Pascal per la prima volta, c'è in appendice, un riepilogo della sintassi, delle funzioni standard e delle parole riservate di Pascal.

GRUPPO
EDITORIALE
JACKSON

Alan Miller

PROGRAMMI SCIENTIFICI IN PASCAL

79